DOCUMENT RESUME

ED 083 825 EM 011 584

AUTHOR	Cowell, Wayne Ed.
TITLE	Proceedings of the Software Certification Workshop, Snow Mountain Ranch, Granby, Colorado, August 27-30, 1972.
INSTITUTION	Argonne National Lab., Ill.; Colorado Univ., Boulder.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	Aug 72
NOTE	160p.; See also EM 011 585
EDRS PRICE	MF-$0.65 HC-$6.58
DESCRIPTORS	*Computer Programs; *Computer Science Education; Conference Reports; *Developmental Programs; Information Dissemination; Material Development; *Mathematics; *Mathematics Materials; Program Development
IDENTIFIERS	Computer Software; Mathematical Computation; Mathematical Software

ABSTRACT

The processes by which statistical software is produced and made available to users were the main concerns of the conference. Six major topics were considered in relation to these software programs which perform the basic mathematical computations required in science and engineering: 1) the quality of mathematical software; 2) education and internships in software evaluation; 3) review of research on testing, portability, and library development; 4) user needs and software program development; 5) publication of mathematical software; and 6) creation of organization to foster mathematical software development. (PB)

ED 083825

Proceedings of the
SOFTWARE CERTIFICATION WORKSHOP

Snow Mountain Ranch
Granby, Colorado
August 27-30, 1972

Wayne Cowell, Editor

EM 011 584

FILMED FROM BEST AVAILABLE COPY

Proceedings of the

SOFTWARE CERTIFICATION WORKSHOP

Snow Mountain Ranch
Granby, Colorado

August 27-30, 1972

The Workshop was conducted as part of a study entitled "Planning an Approach to Testing and Dissemination of Computer Programs for Research and Development," supported by the National Science Foundation under Grants AG325 and GJ31681 with Wayne Cowell, Argonne National Laboratory and Lloyd Fosdick, University of Colorado, as principal investigators. These proceedings were taken from tape recordings of the Workshop as edited by Wayne Cowell.

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

TABLE OF CONTENTS

Page

INTRODUCTORY REMARKS ABOUT THE WORKSHOP - *Wayne Cowell* 1

PRE-CONFERENCE SESSION--Library and Certification Efforts in Great Britain - *Henry Thacher* 4

I. MATHEMATICAL SOFTWARE QUALITY

Comparing Numerical Methods for Ordinary Differential Equations - *T. E. Hull*. 8

Review of NATS Project - *Wayne Cowell*. 26

Validation Procedures for the Boeing Library - A. C. R. *Newbery*. 44

SUBGROUP DISCUSSION OF TOPIC I

- Questions 54

- Summaries

  - *Lloyd Fosdick*. 55
  - *Wayne Cowell* 56
- Draft Certification Statement 59

II. EDUCATION AND INTERSHIP IN SOFTWARE EVALUATION

Design of a Course on Algorithm Testing - A. C. R. *Newbery* 60

Algorithm Editor's Experience with Student Assistants - *Lloyd Fosdick*. . 63

SUBGROUP DISCUSSION OF TOPIC II

- Questions 76

- Summary

  - *Henry Thacher, David Young* 77

TABLE OF CONTENTS (Contd.)

Page

III. RESEARCH ON TESTING, PORTABILITY, AND LIBRARY DEVELOPMENT

Portability Problems and Solutions in NATS - *James Boyle* 80

Unified Standards Approach to Testing - *Walter Sadowski*. 90

SIGNUM 1971 Nonacademic Numerical Mathematics Survey - *F. N. Fritsch* . . 93

An Interactive System for Studying Semantic Models of Computer Programs - *Richard Fairley*. 99

Two Reports on Recent Studies (Abstracts) - *Dorothy E. Lang*. 105

SUBGROUP DISCUSSION OF TOPIC III

Questions 106

Summaries

Leon Osterweil 107

Richard Fairley. 109

IV. USER NEEDS AND MATHEMATICAL SOFTWARE DEVELOPMENT

Technology Transfer Project; Need for Quality Software - *Roberta Smith* . 111

Comments on User/Expert Relations - *Edward Ng*. 115

SUBGROUP DISCUSSION OF TOPIC IV

Questions 121

Summaries

Seldon Stewart 122

William Hetzel 123

V. PUBLICATION OF MATHEMATICAL SOFTWARE

A Journal of Mathematical Software - *John Rice* 126

TABLE OF CONTENTS (Contd.)

Page

VI. ORGANIZATION TO FOSTER MATHEMATICAL SOFTWARE DEVELOPMENT

A Mathematical Software Alliance - *Wayne Cowell* 130

Remarks on IMSL Library Activities - *Edward Battiste* 133

NSF Network, Activities and Plans - *Gordon Sherman* 141

SUBGROUP DISCUSSION OF TOPIC VI

Questions 144

Summaries

Stuart Lynn 146

Charles Lawson 151

LIST OF ATTENDEES 155

INTRODUCTORY REMARKS ABOUT THE WORKSHOP

Wayne Cowell -- The term "mathematical software" has gained acceptance in referring to those programs which perform the basic mathematical computations required in science and engineering. Our concern at this workshop is with those processes by which such software is produced, perfected, and made available to the user community. In addressing this problem, we will ask and seek answers to two kinds of questions. First are questions which are characterized as "what" questions. What are the main issues and problems? What should be done about them? We will make an attempt to identify and isolate the issues which require our attack. The second kind of questions are the "how" questions. How can we organize ourselves so as to accomplish the work we see needs doing?

If you have looked at your agenda, you probably have identified the first four* topics as "what" type questions. The last topic is the organizational "how" type question. I believe that we could meet here and consider the first four* topics and feel that we have done something worthwhile. But if we started with the organizational topic, we would probably find ourselves going off on all kinds of side issues while we tried to understand each other, and worked through some of the ways in which we were using language together. So, it is very important, I think, that we have scheduled an attack on the "what" type questions first. After that we will be better prepared to wade into the question of organization.

The conclusions that we reach here will form a basis for action. I do not mean that we can do detailed planning for a specific organization, but I do mean that the University of Colorado and Argonne National Laboratory are prepared to provide leadership toward the formation of new collaborative structures. But this leadership will be meaningful only if it seeks counsel from the software community as to the direction we should take. An exchange of ideas which leads to this understanding has already begun. It will intensify during this workshop, and will continue thereafter.

---

* A discussion of the agenda followed and resulted in the addition of the topic entitled "Publication of Mathematical Software". Thus the revised agenda has a total of six topics.

As far as the workshop procedures are concerned, we wanted to design a format within which we could proceed in an orderly way. At the same time, we did not want to inhibit any spontaneity or diversity of opinion. So we decided to have informal presentations on each of the topics followed by small group discussion. The presentations will vary considerably in style and length; some will be up to an hour and will deal in considerable depth with some area of importance to us. Others will be brief summaries of recent work.

Although we expect that presenters will wish to answer any immediate questions we might have, we won't try to engage in extensive discussion in the whole group. Instead we will break up into two subgroups. You have received a list of people assigned to groups A and B. In each of these subgroups there is a spokesman for each of the discussion topics. The task of the small group will be to attack each discussion topic using the questions on the agenda. These questions won't necessarily cover everything you want to say but are intended as a guide to your discussion of the given topic. When there has been adequate discussion (this might vary anywhere from 15 minutes to an hour, or longer), we will reassemble as a whole group, and the two spokesmen will give a verbal report of the discussion. They will summarize the highlights and attempt to capture the spirit of the opinion that was expressed. At that point we can have general discussion.

Our hope is to cover the first five topics in about 1-1/2 days to 1-3/4 days. That will leave 3/4 day to one day for the organizational question.

Presentations to the entire group will be recorded. We will transcribe and edit the tapes and will produce proceedings for distribution to everyone here and for use in our study report to the National Science Foundation.

PRE-CONFERENCE SESSION --
LIBRARY AND CERTIFICATION EFFORTS IN GREAT BRITAIN

Henry Thacher -- It is always a temptation to be somewhat parochial in one's views, and to stress the achievements of one's own friends and colleagues. The danger is particularly acute in subjects such as certified software development, since it is not glamorous enough to support as many trans Atlantic commutation tickets as some other areas of computer science and mathematics. I hope, therefore, that you may be interested in some of the information I picked up during a brief visit to Great Britain in July and August of this year. In view of the fact that I made no particular effort at an exhaustive search, the amount of activity in the area which I encountered is quite remarkable.

The first group which I visited was at Chelsea College, University of London, where R. F. Shepherd has a two-year grant for L 13,400 ($33,000) from the Science Research Council. I didn't meet Shepherd, who was taking a long weekend, but had a good chat with John Pemberton, who is the only numerical analyst on the project. He is a recent Ph.D. from Butcher, and is primarily concentrating on differential equation routines. They are strongly committed to Algol 68, and will use it as their primary algorithmic language. There is an additional slot for a numerical analyst which they have not been able to fill.

My next contact was with Mike Powell, who is responsible for the library at Harwell. This is a well-established library for a strong numerical analysis group. They have the capability to write almost all their own routines, and are willing to make their library available to other organizations, provided they are not used directly for financial gain, and are properly acknowledged in publications. Powell gave me a copy of the catalog (Report AERE-R 6919 and Supplement No. 1). A significant feature of this is a tutorial section which, in addition to information on naming conventions and so on, gives specific advice on the choice of algorithms for data-fitting, optimization, and linear algebra.

My next visit of interest to certification was to Nottingham, where I spent most of a day with Brian Ford, and some of his colleagues of the Nottingham Algorithms Group (NAG). As you probably know, this project originated as a collaborative effort among six universities all of whom were to receive ICL 1906A computers. Since then, the project has expanded to include users of several other machine types. Perhaps the most significant characteristic of the NAG effort is the careful coordination, and highly standardized procedures used to ensure uniformity and high quality. I have a copy of the reference manual and of the overall library manual, but not yet of the individual subroutine write-ups. Ford and several of the members of the group are planning to visit North America in the spring of 1973, and would be very much interested in visiting organizations with interests in the area of subroutines libraries and certification, particularly if some contribution could be made toward their expenses.

Finally I spent a day at the Edinburgh Regional Computing Centre, with David Taylor (formerly of Argonne National Laboratory). This centre serves a variety of customers, inside and outside the university. Again, the quality of documentation is heavily stressed. An interesting feature of the Edinburgh library is the fact that it is treated as a partially integrated collection of sublibraries -- SSP, BMD, etc. One of these sublibraries is especially designed for the unsophisticated user with robust routines with minimal flexibility, and minimal demands on the programming and numerical analysis capabilities of the user.

Ediburgh also sponsors some work on program testing. A recent report by Rieger on testing of differential equation routines from their program library includes a battery of about 100 differential equations with known solutions.

In addition to these installations with primary interest in subroutine libraries, I also visited Oxford (Fox), Bradford (Hunter, Dowell), St. Andrews (Curle), and Lancaster (Clenshaw, Kershaw).

In the course of this trip I received several rather strong impressions which may be worth reporting:

1. The state of numerical analysis in Britain appears considerably healthier than in the United States, particularly at the intermediate level. In addition to well-established research-level faculty at most universities, almost all computing centers have several masters level (N.B. This means all the course work for the doctorate) numerical analysts engaged in consulting with users and with library development. There appear to be more jobs at this level than there are people to fill them.

2. Many of the major figures in British numerical analysis (e.g., Wilkinson, Fox, Joan Walsh, Powell) are actively cooperating in development of improved software, although certain others have remained somewhat aloof.

3. The wide variety of installations at which mathematical software is being developed indicates a continuing need for some coordination. The proposal for an IFIP Working Group on Mathematical Software is receiving support from Powell and Ford. Einarson seems to be the spark plug.

DISCUSSION TOPIC I -- MATHEMATICAL SOFTWARE QUALITY

COMPARING NUMERICAL METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS

T. E. Hull - Here's a plan of the talk (TH-1)*. I'd like to spend time on two topics. Most of the time will be spent on the first one, which is a summary of the results obtained for testing methods for solving non-stiff systems of differential equations. This work has been completed in that it has appeared as a technical report, and has been accepted (with modifications) to appear in the SIAM Journal on Numerical Analysis in December. What I will do is proceed through the main ideas of that paper. First, I want to mention the conceptual basis which was important in developing our results because we felt that careful definitions of what we were trying to measure were important. Then I will give definitions of "problems" and examples of them; then "methods," and examples of methods, followed by the criteria we used, and then a summary of our results. Finally, I will mention briefly the work we started as a sequel and area currently working on, viz., testing methods for solving stiff systems of equations. In concluding, I will discuss a few ideas about testing and proving programs. Our long-range goal is to develop programs which are correct (whatever that means) and which are carefully tested so as to support the idea of correctness, and also to enable comparisons among different algorithms.

I will start by showing a slide (TH-2) outlining the beginning of a conceptual basis. We have to begin with the definitions of three things:

---

* The notation XY-n will designate the n^{th} transparency of the speaker whose initials are XY. Copies of the transparancies follow the narrative.

The definition of what we mean by "problem" (and also classes of problems) the definition of what we mean by "method" (and classes of methods), and we must agree on the criteria we will use for the comparison of methods. The first two are the things we need to define if we are going to prove correctness. We would like to prove that a particular method solves a particular problem. If we are comparing different methods on the same problem, we need a comparison criterion as well. We have to say what we mean. For example, we could speak of the average cost -- the one that the person that is paying the money to solve the problem is likely to think of. His objective is to find the method that minimizes the average cost. But another person, for example a man interested in control theory, might be very concerned with maximum cost. His idea of the best method would be the one which minimized the maximum cost of solving problems. (There are analogous situations in approximation theory.) Having made these definitions, we can then talk about one method being better than another, relative to a class of problems, and according to a particular criterion.

When we get down to practical cases we have to name classes of problems and make the appropriate definitions so we know what it is we are talking about. In the case of differential equations, I think this is the most difficult part of the problem and is the only part which brings forth any controversy.

Well, here is what we think is involved (TH-3). First of all you have a system of differential equations. We have a function, f, and we need initial conditions. Also, we need to know how far we want the integration to be carried. You could continue until some component exceeds a particular value. We chose the simplest one: Continue until the independent variable reaches a certain value. That completes the problem specification. What the user would like to do is to specify a tolerance, τ, per unit step. But this can be interpreted in a number of ways, and you have to be specific. Of the two methods on the slide, the most elegant is the second one. This is the one we would use if we were trying to prove the correctness of the method. We prove that we have solved a problem that does not differ significantly from the original problem. That is quite elegant and fits in quite well with the theorems we prove about correctness.

For testing programs, we want to test to see if the method solves the problem. So we use 1, which is almost equivalent to 2, but, for practical purposes, it is much more useful than 2. We will define the local error to be the difference between the computed result at x_{i+1} and the value of the true solution through the computed result at x_i. It must be bounded in norm by τ times the step size. In our experiment we require that the method produces a sequence of values with the property that at each step the local bounds are satisfied.

The user would like to specify global error requirements. But all the methods we tested are step-by-step methods which try to keep the local error down. It may be that the global error is very large, or very small, but this depends on the differential equation, not on the method. So, for that reason, we think that, for step-by-step methods, the proper criterion is to insist on their trying to solve the problem as specified above, and to compare the costs they incur in solving that problem.

Finally, we think that in specifying the problem you need something about the scale of the problem. The user is responsible for saying something about the scale of the problem, and we felt that the user could most easily provide a bound, h_{max}, on the step-size.

At the bottom of the slide we list these six things, put brackets around them, and announce that a problem is a sextuple! (This list then also becomes the calling sequence of the subroutine.)

For testing purposes, we specified five classes of problems. I will just give a few quick illustrations here. You can see the full list in the published paper. What I have listed (TH-4) is some of the functions. The final value is always taken to be 20. We tried three tolerances with each of the equations. There are five classes. We took five fairly different classes, our idea being that a method might turn out to be good at one particular class of problems but not as good at another one. (This turns out to have a happy ending -- we found that methods that were good in one class tended to be good over all classes. The first class was just a class of five single equations. (Everyone has to test the first one!) The next class was a collection of small systems. The system shown is supposed to represent the interplay between predator and prey. It is a very hard system to solve -- many methods cannot go over three or four cycles. Then class C contains moderate systems, including a five-body problem in the last example. Then we have five very simple elliptical orbit problems. Finally, we took a collection of well-known, second order differential equations (like Bessel's equation) and converted them to systems of first order equations.

Now we come to methods (TH-5). We think of a method as being characterized by four things, illustrated by the famous Runge-Kutta method. We have: (1) a formula for calculating, (2) a formula for estimation of error, (3) an acceptance criterion, and (4) a strategy for choosing step size. The program first of all decides what it is going to do; it decides on the step size, perhaps the order of the method in some cases and so on, it takes a step. Then it makes an estimate of the error. Then it decides what to do with regard to accepting the step, etc. (The programs are well structured, so you can see what they are doing.)

Now I list the methods we tested (TH-6). We tried some Runge-Kutta methods. We are convinced that variable order methods are the only multistep ones worth testing, since the "best" method will be a higher order method if you make the tolerance extremely small. The method must be able to adjust its order to suit the tolerance. We have tried other methods as well. I should point out that we are as interested in setting standards as we are in finding the best.

Now we come to criteria (TH-7). We felt that in the end it was the cost of solving the problem that really mattered, and we were thinking of computer time rather than space. So we used computer time -- mainly we used seconds on a 7094 because we found that we couldn't measure time as accurately on the 360. We ran all of these tests a number of times on different machines and under different circumstances. The thing we soon learned was that it was important to measure two components of the time: What we call the overhead time and the time spent evaluating functions.

Although we like the conceptual basis of having collections of methods solving problems, the fact is that in practice the methods do not actually solve the problems exactly as specified, and we found we had to keep track of how well methods did in solving. For instance, we counted the number of times a method deceived itself, that is, the number of times it actually took a step, and accepted that step, thinking that the error was less than tolerance, when in fact it wasn't. We also kept track of the maximum error in all the times it deceived itself. Usually if a method decieved itself only 2% or 3% of the time, the worst it ever did was be off by a factor of 3 or 4 in tolerance.

One of the difficulties we ran into, which seems more typical of differential equations than, say, with matrix calculations, is that there are so many different effects. For example, the question of starting was one that bothered people at the Bell Labs quite a bit. They found that the starting step size could have quite an effect on the cost of solving the problem. Their solution was to take four of these and average over them. Since our runs were not terribly long, the starting difficulty could not be ignored. We decided to let each method find its natural step size for starting, then we turned on the clock, set the calcualtion back to X_0 and allowed the method to take step, and go from there. We felt that this tended to minimize the effects of starting. We do not think starting is unimportant, but we think we should have another measure of how good the method is at starting. We minimize effects of round-off by just doing everything in double precision, and making our tolerance no smaller than 10^{-9}. We avoided stiffness by simply not having any problems that were really stiff. We had some that were slightly stiff. We also avoided discontinuities. We were trying to compare methods according to their ability to chug along, without saying much about their ability to cope with stiffness, continuity, or starting. Otherwise we would have a result which was merely an average of a whole lot of effects.

Finally, our results (TH-8). Varying the order seemed to be important. Variable order methods seemed to be able to do well over a range of problems.

Except for that fact, the results were amazingly uniform. They were not only uniform within each problem class, but seemed to be between different classes. In our paper we summarized the results, and here is a part of our summary. We felt that an average over the classes was representative of what these methods were capable of doing, so long as we calculated averages separately for each tolerance.

Notice the Runge-Kutta overhead time. It is pretty competitive with the Bulirsch-Stoer in total time also. Although it seemed competitive in this one example, we did not rate it very highly. It's maximum error is 142 times the tolerance which is not good compared to what other methods were able to do at a few times the tolerance. Also, this is its best tolerance.

This is typical of the Runge-Kutta methods. They tended to do well for one tolerance but not others. So, as general purpose methods, we felt they could be set aside. Now we come to the two interesting ones: Extrapolation and variable order Adams. Total time for the latter was certainly longer than total time for Bulirsch-Stoer. The interesting thing is that Bulirsch-Stoer is quite a bit better in overhead, but on the other hand, it took many more function calls, more than twice as many. You see clearly that you have to separate those two costs. Which method is best is going to depend on how much its going to cost for function calls. You can say that Adams-Krogh is spending a lot of time caompared to Bulirsch-Stoer in deciding how to avoid making function calls, and it only pays off if the function calls are costly.

Our conclusions (TH-9) are: (1) the extrapolation method is best if the functions are simple, but (2) variable order Adams is best if the functions are complicated. We went to the trouble to see if we could find the break-even point between the two. We looked at the right hand sides, and came to the conclusion that you could count up the number of arithmetic operations per component (or their equivalent). For example, if there is a sin x on the right hand side, count it as roughly 5 arithmetic operations. Then, 25 arithmetic operations per component is about the break-even point. So, if you have less than that, your best method would be extrapolation, but if more than that, variable-order Adams. This is pretty rough and it actually ranged between 5 and 45. Finally, the low order Runge-Kutta methods still seem to be competitive if the functions are simple and the tolerance is not very small. (In the language of James Lyness, the complexity of the functions is an important feature in the "performance profile" of a method; stiffness is another such feature.)

I have two more topics I would like to spend some time on.

We are currently testing methods for stiff systems (TH-10). We decided that it would be a good idea to get real-life problems. We have a collection of 30 or so of them, which we sent to some people we knew were interested in this area. Quite a few people sent back comments. These problems have all been coded and I think they are now ready. Some of the problems are quite large, and we had to do some over-laying because a couple of them were so large that they just occupied too much of the memory of the 370.

The program we used in testing is called DETEST. It stands for "Differential Equations Tester." (We thought we had found the right name; it's important to have the right name!) DETEST has been re-written and it is going to be published as a Technical Report. It has been extended so that it collects many different statistics. The criteria are more complicated for stiff problems. One of the most difficult things of all is what to do about the transition from the stiff region to the non-stiff region. At the moment, what we have decided to do is to measure all the statistics over the whole problem and also to measure them separately from a point, that we can determine, at which the stiff components have all died out. We selected a point, which we arbitrarily set for each problem, and we go back and restart the integration there, and see how well it does. In the region where the stiffness would slow down more usual methods, these new "stiff" methods really show their capabilities.

Now I come to the last topic. I will try to be brief. I think we should consider very seriously the question of correctness. It is a big topic. All I can do is suggest a couple of ideas. What do we mean by saying a program is correct? (TH-11) I like to view it this way: The program correctly evaluates the prescribed function. And then the question comes down to: How is that function prescribed? Here are some examples of the things we have to keep in mind. Think of the problem of solving a linear system, Ax = b. Suppose we imagine a program being carried out in exact arithmetic. Then we can think of the function that is being evaluated as indicated in 1. In floating point arithmetic, the mapping is defined in a different way as shown in 2. There are other alternatives for example as shown in 3 and 4, but in all cases we have some mapping in the back of our minds and require a proof that the program evaluates that function correctly.

We could consider a number of things (TH-12). When we are testing programs, I think we should think in terms of testing that is associated with proving correctness of the program, and also testing that is associated with comparing efficiency. Under correctness I think we can certainly insist that the program be correctly coded. We should encourage the use of structured programs, so that the program will not only be correct but be seen to be correct. Testing lends support to this. (The standards of rigor are much higher in our profession than in mathematics.) We can also consider the programs being correct in a different sense, in the sense of a backward error analysis. You still need the structured program to organize the proof of correctness. Sometimes it is helpful to use assertions. So we can view correctness in different senses If you have proven a good theorem in 2, in a way you don't really need 1. For efficiency, you can simply take a number of programs that are to solve the same problem, do a lot of testing, and compare relative efficiency. I think we need to clearly define problems, to clearly define methods, and criteria. I think we have to be clear about whether we are proving correctness in this sense or that sense, and what it is we are using our testing for. The emphasis in testing is certainly very dependent on the particular application area.

In the case of ODE's, I have been primarily discussing 3. I do not think we know much about 4. We know about the relationship between the order of the methods and the tolerance a little bit, but we do not really know any theorems. So, I have been emphasizing 3 all along. After we had done all our testing, we decided that 1 was terribly important, and we started to restructure our programs. So we are now doing 1 and we have been helped in this way with 2.

I want to show you a theorem about the correctness of a program for differential equations (TH-13). We have a class of problems, as shown. We assume no round-off; that enables us not to have to put all kinds of restrictions on. We can prove an effectiveness theorem that guarantees that the method solves any problem in the class. Thank you for your attention.

PLAN

Summary of results obtained with
methods for non-stiff systems:
- conceptual basis - p, m, c
- problems
  - examples
- methods
  - examples
- criteria
  - examples
- results
- conclusions

emphasis here
is on comparisons

(to appear in SIAM J Num Anal,
Dec., 1972 - Hull, Enright, Fellen, Sedgwick)

Mention of current work on methods
for stiff systems

Mention of a couple of ideas on
proving correctness of programs

Slide TH-1

We need to agree on three
definitions:

(1) Definition of problem, say p,
and on class of problems, P.

(2) Definition of method, say m,
and on class of methods, M.

These two definitions are all
we need for "correctness" -
we would try to prove, e.g., that
a particular m solves any p
in a particular P.

(3) Comparison criterion c(m,p);
also C(m,P), e.g. average, or max.

Then we can say m is better than
m', relative to P, according to C,
if C(m,P) < C(m',P).

Also m is best in M, if ... .

Slide TH-2

Specification of a problem

$y' = f(x,y), \quad y(x_0) = y_0$

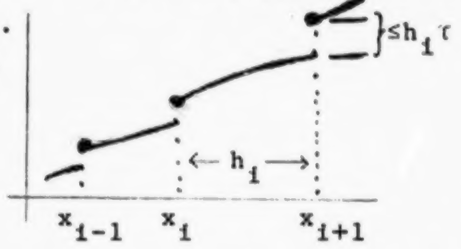
also, e.g.,

x_f - the final value of x

τ - the tolerance per unit step

e.g.

(1)



(2) $\exists z(x) \ni$

$z(x_0) = y_0, \quad z(x_f) = y_f$

and

$\|z' - f(x,z)\| \leq \tau$

h_{max} - the max. step-size

So $p = \langle f, x_0, y_0, x_f, \tau, h_{max}\rangle$

Slide TH-3

Problems f's below; x_0, y_0 not shown

$x_f = 20, \quad \tau = 10^{-3}, 10^{-6}, 10^{-9}, \quad h_{max} = 20$

A1 $y' = -y$

2 $y' = -y^{3/2}$

⋮

5

} single equations

B1 $y_1' = 2(y_1 - y_1y_2)$

$y_2' = -(y_2 - y_1y_2)$

⋮

5 Five body problem

} small systems

C1 $y_1' = -y_1$

$y_i' = y_{i-1} - y_i, \quad i=2,\ldots 9$

$y_{10}' = y_9$

⋮

5 Five body problem

} moderate systems

D1 Orbit problem $\varepsilon = .1$

⋮ $.3$

⋮ ⋮

5 $.9$

} Orbits

E1 Bessel, van der Pol, Duffing, etc.

⋮

5

} higher order reduced to first order

Slide TH-4

Example of a method

A Runge-Kutta method based on

$$\bar{y}_{i+1} = y_i + \frac{1}{6}(k_0+2k_1+2k_2+k_3)$$

where $k_0 = hf(x_i, y_i)$

$$k_1 = hf(x_i+\frac{h}{2},\ y_i+\frac{k_0}{2})$$

$$k_2 = hf(x_i+\frac{h}{2},\ y_i+\frac{k_1}{2})$$

$$k_3 = hf(x_i+h,\ y_i+k_2)$$

Method consists of:

Calculation of y_{i+1} by taking two steps of size $h_i/2$

Estimate of error $\text{EST} = \left\| \frac{y_{i+1} - \bar{y}_{i+1}}{15h_i} \right\|$

where $\bar{y}_{i+1}$ is from one step of h_i

Accept if EST ≤ .9τ

Choose

$$h = \begin{cases} \min(h_{max}, x_f - x), & \text{on entry} \\ \min(.9(\frac{\tau}{\text{EST}})^{1/4} h_{old}, h_{max}, x_f - x) & \text{otherwise} \end{cases}$$

Slide TH-5

Methods tested

Several Runge-Kutta methods of orders 4, 5, 6, 8

Several variable-order, variable-step Adams methods due to Krogh, Gear, Sedgwick

Extrapolation method due to Bulirsch & Stoer

Others only partially.

Slide TH-6

Criterion (from DETEST)

We used computer time (in seconds on a 7094-II) - but it was necessary to measure two components separately, namely

(1) overhead time

(2) time spent evaluating functions (we actually measured total time, & also counted the no. of function calls)

However, the methods didn't quite solve the problems, so we also measured

(3) number of deceptions (in percent)

(4) max. error (in units of τ)

Note: We suppressed effects of starting roundoff, stiffness, discontinuities.

Slide TH-7

Results

Varying order certainly important for such a wide range of τ.

Otherwise, results were very consistent - not only within each problem class A, B, ..., E, but also between different classes.

We can therefore summarize over all of A, B, ..., E for each τ.

E.g., with $\tau = 10^{-6}$, we get

	OVHD	FCN CALLS	TOTAL TIME	PERCENT DECEIVED	MAX ERROR
B-S	37.1	26704	47.7	0.7	2.3
Krogh	59.1	11353	63.1	1.4	7.3
RK6	41.1	23540	48.8	1.6	142.5

SECS

IN UNITS OF τ

Slide TH-8

Conclusions

(1) Extrapolation is best if the functions are simple, but

(2) Variable-order Adams is best if the functions are complicated.

(3) The "break-even" point between B-S & Krogh is, roughly, when about 25 arithmetic operations (or their equivalent) are required per component.

(4) RK4, RK5 are competitive when the functions are simple and $\tau \approx 10^{-3}$.

Two parameters for "performance profile"
- function complexity
- stiffness

Slide TH-9

STIFF SYSTEMS

Currently testing methods for stiff systems

(1) We have a collection of "real-life" problems.

(2) We are programming 4 methods
- Gear's method
- Implicit Runge-Kutta
- Trapezoidal with extrapolation
- Enright's method.

(3) Criteria more complicated
- must count Jacobian evaluations L-U decompositions, etc., as well as function evaluations & overhead. And what about transition from stiff to non-stiff?

Slide TH-10

CORRECTNESS

What do we mean by saying a program is correct?

I like: it is correct if it correctly evaluates the prescribed function.

For examples, consider prog. for $Ax = b$ and the following alternatives:

(1) Exact arith:- $A,b \rightarrow y \ni Ay = b$, provided ...

(2) Fl. pt. arith:- $A,b \rightarrow y \quad (A+\delta A)y = b$ with $|\delta A| \leq 1.01(2n^2+h)\rho\|A\|u$, provided ...

(3) Fl. pt. arith:- $A,b \rightarrow y$ $\|y - A^{-1}b\| \leq \sim\!\sim$, provided A ...

(4) Any arith:- $a,b \rightarrow y$ that is obtained by following LU decomp., etc.

A special case of (4) is to show the program is correctly coded.

Slide TH-11

We could consider:

Correctness

(1) In sense of being correctly coded
- use "structured" programming (program must be seen to be correct)
- testing lends support

(2) Error analysis theorems
- sometimes use assertions
- testing lends support

Efficiency

(3) Testing for comparisons

(4) Theorems
- testing lends support

Need to be clear about what we are trying to do - e.g. p, m, c

Emphasis above will depend on area - lin. alg., functions, o.d.e.'s, etc.

Slide TH-12

Effectiveness theorem:

If $P = \langle Ay \mid \|A\| \le 1,\ x_0,\ y_0,\ x_f,\ \tau,\ h_{max} = 1\rangle$
$m = \langle$2 Kutta steps, EST, $\le .9\tau$, $h = \sim\rangle$,
no roundoff
then m solves any p in P, in
sense that m produces y_f
$\ni \exists\ z(x)$ where $z(x_0) = y_0$
$z(x_f) = y_f$
and $\|z' - f(x,z)\| \le \tau$.
(result is best possible)

Also have results for stiff (Enright)
& for non-linear (Sedgwick)
Also for Adams (Sedgwick)

Slide TH-13

REVIEW OF NATS PROJECT

Wayne Cowell -- Because the NATS project has been reported at several meetings lately, some of you have seen the slides I have and are aware of what we are doing. I will not go into too much detail here but will give a little different emphasis in a few places.

There is a term which is gaining some currency at Argonne which I would like to share with you. For budget purposes and our own understanding, we have been carefully documenting our research activities in computational mathematics and have come up with the idea of a systemetized package of computer programs which we have defined as shown here (WC-1). Of course, an example of systemetized collection is EISPACK (WC-2). As many of you know, Edition 2 is now being distributed from the Argonne center as a certified package. It's a collection of 34 routines to solve certain cases of the eigenproblem and is available for IBM 360, CDC 6000-7000, PDP-10, Univac 1108, and Honeywell 635. Edition 3 is in the hands of the test sites, which we will list in a moment. Concerning the point (in WC-1) about minimum concern for systems details, the IBM version also has a control program called EISPAC (without the K), which enables the user to describe his eigenproblem at a very high level. Jim Boyle is the author of that.

Another collection which we are in the process of systemetizing in the NATS project is the special function package. It is indicated on the slide (WC-3) how we stand. Most of this is Jim Cody's work. We will soon be announcing the availability of the exponential integrals for 3 machines: IBM, CDC, and Univac. Cooperating with us on this part of the work are the University of Wisconsin, and the University of Texas.

I want to list the other field test sites which have been involved with the NATS project (WC-4). This is a project to test, certify, and disseminate mathematical software. At this stage, it is not an attempt to build up a comprehensive library, but rather an attempt to produce systemetized packages and thus to learn how to produce them, to learn something about the kinds of people involved, the time it takes, and what it costs.

The procedures that we follow are roughly these (WC-5). The routines are prepared at the principal institutions, in this case, Argonne, University of Texas, and Stanford. They are then field-tested at the cooperating sites, certified, distributed, and supported. I want to say just a bit more about what some of these terms mean. Field testing involves two parts (WC-6). The first part is checkout at the computer center. We supply a tape with routines, test cases, and drivers. We have assembled about 80 test cases for EISPACK and drivers which print out the norm of the residual. That information comes back to us for evaluation. When the routine is ready to be made available to test-site users, there is effort at the test site to offer consulting services, to make it very easy for people to use these routines, and to feed back information on users' experience. This is a little harder to organize than computer center checkout. Scientists have to be shown that using the new routines is going to be to their advantage.

What do we mean by certification? I think this focuses very directly upon the topic we are considering at this workshop. The meaning of certification was actually a subject of a great deal of thought and discussion on our part. We can't mean, of course, that a certified code is guaranteed in the sense that we pay for the computer time if it doesn't work! The idea which finally emerged is akin to the scientific tradition of saying that the tests (experiments) we have performed can be duplicated. We will furnish you with all the information we have available and you can run the same tests. Also, we risk our reputation; we stand behind what we have done. To express this idea, we have included this statement (WC-7) on each of the routines which we regard as certified. We could say this sort of thing without any testing at all, but that would be foolhardy!

We have arrived at some guidelines which I have written down here as principles. These first 3 principles express requirements that a collection of routines be considered as a candidate for systematization (WC-8). Without these remaining principles (WC-9) you couldn't really organize a good testing program. First of all, we would insist that computable measures of quality exist. In the case of EISPACK we have used the norm of the residual stemming from Wilkinson's backward error analysis. In the case of the special function codes, I think most of you are aware of Jim Cody's methods for statistically comparing the performance of a routine with the performance of multiple precision calculations. Finally, the last statement is that the routines have to satisfy a basic need - there is a demand for them. Because there is a demand there is also a potential for organizing a collaborative effort, and we believe that is necessary for this type of work.

This diagram (WC-10) gives an over-view of the project. To illustrate we might follow EISPACK through the diagram. Assembly into packages, preparation of test cases and preliminary testing took place during the Spring of 1971. I might say that the use of the Argonne RESCUE system was extremely valuable. Jim Boyle will have more to say about that later on. Edition 1 went to the test sites in the summer of 1971. The test sites responded in various ways, most of which were very gratifying. Besides computer center checkout, users guides were prepared in several places. A comprehensive one was prepared by Yasuhiko Ikebe at the University of Texas. The University of Chicago published an EISPACK newsletter, and made available a good deal of consulting. The control program was developed for Control Data computers at Northwestern University. Interface routines were written at Stanford, which enabled users to call routines using familiar calling sequences. So far as we can tell, there were a couple of dozen users (besides the computer center checkout) in various fields--economics, geo-physics, chemistry, astronomy, nuclear engineering, mechanical engineering, statistics. [illegible] have case histories on some of these. Also, the routines were used in an engineering summer conference at the University of Michigan both in 1971 and 1972.

In December 1971, there was a meeting of test site representatives at Argonne in which we reviewed the progress that had been made on the field testing, and obtained feedback from it at that point. We made one key design decision. The field test representatives felt it wrong to have machine-dependent parameters in the calling sequences. These parameters were taken out of the calling sequences and embedded as lines of code. This resulted in our going from two versions of the package to four, the four differing very little.

Edition 2 went to test sites in March, 1972 and was announced as a certified package in May, 1972. Here's a word about the availability of the package (WC-11). It is available from the Argonne Code Center for those 5 computers. From May 24 to August 9, some 34 requests for the IBM and 28 requests for other computers had been received, and processed. There have been requests since. Each of these actually constitutes an installation. The Code Center does not send to two different persons using the same computer at the same location. So I would say that there are at this time roughly 75 installations across the country which are (potential) users of EISPACK. I cannot say it has been installed and is working at all these.

Let me conclude with a word on costs (WC-12). Here I refer to the cost of creating certified code from published Algol. There are various ways of looking at costs. We could look at it in terms of the cost per Fortran statement. I'm sure you recognize this would be a very rough ball park kind of estimate. We have versions for 5 machines and I would estimate that the cost is about $20 per statement of certified code. I don't want to put Ed Battiste on the spot but I have discussed that informally with him, and he didn't say we had been especially wasteful. We could also distribute costs over the number of users. I think it's very conservative to say that EISPACK will be used eventually by 100 installations. You could distribute a half million dollars over those installations at $5,000 per installation. You would probably have to pay a graduate student more than that to code up Wilkinson's Algol, and the results would not likely be comparable.

But the way of looking at the costs that I like best (and which Jim Wilkinson has reflected upon) is that this work is a step that is necessary in order to take a tremendous intellectual investment over many years by first class people and make it available as practical software. It is very difficult to put dollar figures on that kind of thing. At the Boston meeting where I reported on this, Joe Traub suggested another view of cost, which I haven't really explored any further. He asked if we had any way of measuring the number of replacements, the number of places in which EISPACK has forced out another collection of eigenproblem codes.

Well, those were the remarks I wanted to make about the NATS project. I would like to postpone general questions for later and go on to some remarks by Chris Newbery about validation of the Boeing library.

Slide WC-1

A collection of computer programs will be said to be a systematized package if

(a) each program in the collection is accurate, efficient, fully documented, thoroughly checked, readily available, and effectively maintained;

(b) the collection contains routines to solve a wide spectrum of related problems, reflects the latest techniques, and is organized so as to require a minimum of concern for those computer system details which lie outside the scope of the user's primary interest.

Slide WC-2

EISPACK

Edition 2

34 routines to solve the standard eigenproblem for real general, certain real tridiagonal, real symmetric, real tridiagonal symmetric, complex general, and complex hermitian matrices.

Edition 3

40 routines to solve above problems plus standard eigenproblem for linearly-packed symmetric matrices, band symmetric matrices, and the generalized eigenproblem $Ax = \lambda Bx$ where A and B are symmetric and B is positive definite.

Slide WC-3

SPECIAL FUNCTION PACKAGE

For IBM 360, UNIVAC 1108, CDC 6000-7000 except as noted

Work almost finished

Exponential integrals
Complete elliptic integrals of the first and second kind

Work well underway

Psi function
Bessel functions K_0, k_1
Dawson's integral

In progress

$$\left.\begin{array}{l}\text{Error and complimentary error function}\\ \text{Gamma and log gamma functions}\end{array}\right\}\text{ Univac and CDC only}$$

Fresnel integrals
Riemann zeta function
Coulomb phase shift
Chi-squared integral
Regular Coulomb wave functions
Bessel functions J_0, J_1, Y_0, Y_1

Slide WC-4

NATS is a prototype effort to

TEST, CERTIFY and DISSEMINATE

Mathematical Software

Principal Institutions

Argonne National Laboratory	IBM 360
The University of Texas at Austin	CDC 6600
Stanford University	IBM 360

Cooperating Test Sites

Iowa State University (Ames Laboratory)	IBM 360
National Center for Atmospheric Research	CDC 7600
Northwestern University	CDC 6400
Oak Ridge National Laboratory	IBM 360
Purdue University	CDC 6500
Lawrence Livermore Laboratory	CDC 6600-7600
The University of Chicago	IBM 360
The University of Kansas	Honeywell 635
University of Michigan	IBM 360
The University of Toronto	IBM 370
The University of Wisconsin	Univac 1108
Yale University	PDP-10

Slide WC-5

NATS Routines are

1. Prepared at the principal institutions
2. Field tested at cooperating test sites
3. Certified
4. Distributed from the Argonne Code Center
5. Supported by the developers

Slide WC-6

FIELD TESTING

1. Check-out with prescribed data using testing aids supplied by coordinators.

2. Monitored use by scientists and engineers on their problems.

A CERTIFIED routine is a SUPPORTED routine which has been thoroughly TESTED.

The documentation of each NATS routine includes the following statement:

> This routine has been tested on and is herewith certified for the following computer systems and working precisions.
>
> (List of test sites, machines, operating systems and working precisions.)
>
> The NATS project fully supports this certified routine in the sense that detailed information on the testing procedure is available and reports of poor or incorrect performance on at least the computer systems listed above will gain immediate attention from the developers. Questions and comments should be sent to
>
> (Name and address of contact.)

Slide WC-8

PRINCIPLE A

The algorithms underlying the routines have a sound numerical basis.

PRINCIPLE B

The routines are written in a widely-used source language. They have undergone testing for efficiency and absence of gross errors. Basic documentation exists.

PRINCIPLE C

The routines have been organized into a coherent collection which solves a class of problems.

Slide WC-9

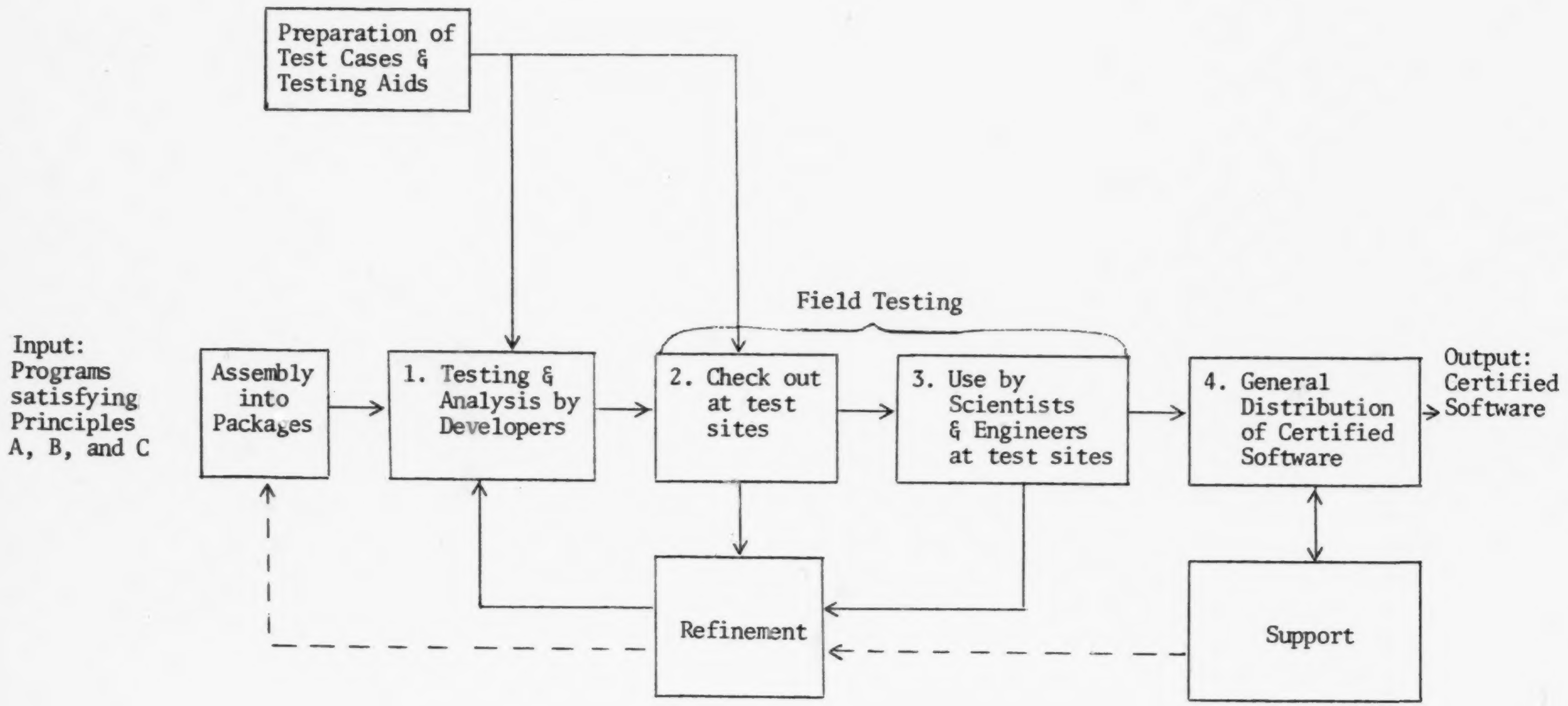
PRINCIPLE D

Computable measures of quality exist.

PRINCIPLE E

The routines satisfy a basic computational need and are required for use in a number of institutions. They have potential for becoming an authoritative standard within the computing community.

Slide WC-10


Preparation of Test Cases & Testing Aids
Field Testing
Input: Programs satisfying Principles A, B, and C
Assembly into Packages
1. Testing & Analysis by Developers
2. Check out at test sites
3. Use by Scientists & Engineers at test sites
4. General Distribution of Certified Software
Output: Certified Software
Refinement
Support

Over-all View of Collaborative Software Testing

Slide WC-11

AVAILABILITY

The second edition of EISPACK for

IBM 360 (including control program EISPAC),
CDC 6000-7000, Univac 1108, PDP-10,
Honeywell 635

is available from the

ARGONNE CODE CENTER.

During period May 24-August 9 there were

34 requests for IBM
28 requests for others

Slide WC-12

COST ESTIMATES (As of August, 1972)

(after original research)

EISPACK in 5 Macine Versions -

$20 per Statement of Certified Fortran Code

or

EISPACK used by 100 installations -

$5000 per installation

or

EISPACK as the step necessary to make a very large research investment pay off in practical software - incremental cost: small.

VALIDATION PROCEDURES FOR THE BOEING LIBRARY

Chris Newbery -- The Boeing library went into business in 1969. It was a fairly large-scale project, about 10 man years, 300 programs, about 24,000 object deck instructions. We were very concerned with validation, and I want to talk a little about our procedures, how we achieved what we did achieve. It's been possible since then to determine how good or bad it was - I'll let you judge that.

In the first six months after we went on the air, they found six bugs in the whole works, and since then they haven't found any. This has not been under conditions of testing by hostile people like yourselves; it's just been ordinary testing on the shop floor in Boeing and at the University of Washington. I'd say there were two reasons for the stress on validation. From the point of view of the top management the reason was simply flow time. It's sometimes thought that aircraft can crash as a result of computer program errors. I'm not sure that's true, I'm inclined to doubt it, but certainly lots of other unpleasant consequences can ensue. If you write the critical path network for production of an airplane all the way from initial design through marketing, the computers occur on the critical path at several places. They occur very markedly in the early design phase, particularly the stress analysis phase, and they occur later in the automation of the machine tools, i.e., constructing tapes which will guide a machine to cut the air foils and cut the various components for the prototype and for the production models. Finally, they are on the critical path again when it comes to analyzing the flight data after the prototype has already flown with lots of gadgets on it measuring stresses and vibration frequencies and temperature changes and so on. Those are the three parts or stages when the computer is on the critical path, and any foul-up holds up the whole project. That can cost thousands of dollars. So that was why the management was kind of up-tight about validation.

The other reason for validation stemmed from our own strictly computerman's point of view. We were concerned with validation mainly because we knew we could never get our product marketed unless we had a good record this way. There are lots of inferior programs available (users groups and so on, to name one of them) and we felt that unless we developed the reputation for being pretty nearly infallible, nobody was going to use our work, and it wouldn't do much good if nobody used it. It's exceedingly difficult to get a man to give up using a program he is familiar with, and get him to switch to a program which works. In the course of this, I've written out a little check list of things which I consider important in the design of validation experiments (follows narrative).

The first one is concerned with being really clear about what your objectives are, what you are testing for. The typical programmer, even an experienced one, tends to be very vague about this. So I think one needs to be very clear of one's objectives and the first commandment, so to speak, says make a list.

The second item is concerned with recognizing that there is inevitably a lot of noise in most of the things one is testing for. Most of the questions you are asking when you test can not be answered by a single test. A few of them can, but I've given an example in the attached sheet where you have to make a large number of tests before you can be reasonably sure that your program is performing satisfactorily in that respect. I could give a different example of the same general kind. Let's say you've got a linear equation solver and you're concerned about whether that equation solver knows when it is beaten. If you give it a problem, which is numerically singular, does it recognize this and go into an error return, or does it just plow ahead? I don't think anybody, however good a programmer or however knowledgeable a numerical analyist he may be, can write a program that is really perfect in that respect. I think however good you are, you are not going to be perfect. For example, if you are going to try to persuade a computer to recognize a numerical singularity, you are going to compare some vanishingly small quantity with some kind of standard. You are going to look at a succession of pivots and compare with some small number and decide to quit when it is effectively singular. Or if you are interested in finding a certain degree of slowness in convergence, you will use that as a measure of whether the thing was effectively singular. If you are looking at pivot sizes and you did not compute your pivot size with perfect accuracy, sometimes the criterion which you prescribed will work the way you intended and sometimes you will be unlucky. Then you will want to know whether the thing is fail-safe; i.e., if it had made errors, would they be errors of the sort that said the system was numerically singular, when in fact it wasn't? But you must not make errors of the opposite kind where it uses garbage and says it is O.K.

The third item is checking to see that the program automatically avoids any readily avoidable numerically ill-defined situation. Scaling of matrices is one example. But I think a more dramatic one, although it is less commonly mentioned, is interpolation where you interpolate on half a dozen points, say in the range from a thousand to a thousand and six. You are going to be evaluating your Lagrangians for very large arguments. You would need to shift the origin to minus 3 to plus 3, do your interpolation, and re-interpret back again. A good computer program will do that. One should check to see whether a program is good in that respect.

The fourth item is concerned with relating the results that you get to some kind of acceptable standard. Very few of us could say whether preserving five figures out of eight on inverting a six by six Hilbert matrix was a good result or a bad result. So I much prefer the result that says the errors were never worse than 1% of the error bound that Wilkinson gives on page such and such. Then you are relating the errors that you observed to error bounds that are mathematically established, and that takes into account the condition number of the problem, and it saves you a lot of mental arithmetic and enables you to relate what you have observed to what you think you should have observed, or relate it to the worst possible case that could have occurred.

Item five is a statement really very much like Hamming's "insight-not-numbers" statement. It says pretty much the same as he says only he says it better. It's also related to Lyness's performance profile concept. It seems to me that if you're evaluating the performance of a matrix inverter or linear equation solver, one would put out the results in some kind of a grid form which lends itself to interpolation. It seems to me that you should vary condition numbers from very easy to what you call tough in your word length. At the same time, in the other dimension, you would want to vary the order of the matrix. See how it behaves on larger matrices. Then you have got large well-conditioned, large ill-conditioned, small well-conditioned, small ill-conditioned. When you have got that, you haven't just got the twenty-four numbers that are put out, you have really got information about the whole two-dimensional continuum of numbers. You can interpolate because those tables should be fairly smooth. And if they are not smooth, then you want to know why, because something has gone wrong. Each entry in your grid will not be the result of a single experiment because that single experiment of course could be non-typical and you must average a large number of experiments to produce each of the twenty-four outputs.

Item six is concerned with consistency tests. This is perhaps a bit more controversial (there are those who dislike consistency tests), but such tests do serve the purpose of enlarging the range of tests that you can give. It is not necessary to know the answer to a problem in order to use the problem for test purposes. Thus you can check consistency in differential equation solvers as long as you know something about the property of the solution, like it is periodic. You do not need to know the whole solution, you can check on periodicity. It is a check that can be misleading, but it is about the best you can do. Likewise there are a lot of consistency tests you can use with polynomials such as seeing whether it thinks p(2x) is a harder polynomial to solve than p(x) or seeing whether it thinks reversing the coefficients, thereby reciprocating the roots, takes more iterations or gives less accuracy. Any substantial difference in its reaction to p(x) and reversed p(x) is an indication that it did one of them wrong.

Item number seven is concerned with testing separate parts of, say, an eigensystem package, in which eigenvalue calculation is followed by eigenvector finding. I think it is appropriate to test those two things separately, because otherwise it gets very difficult to locate what the source of the error was. You can attribute it to one aspect when in fact it resulted from another. And there are, as I mentioned, some situations where you should not do that.

And lastly, (item eight) one should try to break the program to see where its breakpoint is, and to see whether it knows it is beaten when it is beaten. The last sheet that I have given out is a little attachment, and I must apologize for the attachment being somewhat scruffy. In fact, it was not really intended for distribution at this meeting or any other. It is an exact reproduction of little notes I gave to my programmer when I was asking him to test things out. It is an authentic set of tests that I asked him to execute. It does illustrate two points that I think I have mentioned. It deliberately creates Jacobians which will have rank difficiencies of one, two, three and that is a fairly severe test for any non-linear equation solver which uses Jacobians for solving linear equations. The other thing it exemplifies is the performance profile. It tells the programmer to put out a grid, each point on the grid being the result of 20 experiments, so that when you read it, it can be interpolated. So those are my 8 commandments. I would be very glad to hear any comments, criticism, and particularly additions.

A Check-List for Validation of Programs

When one is designing a sequence of experiments with a view to determining the capabilities and limitations of a given computer program, the following check-list may be found helpful.

(1) Make a list of the questions which the tests are supposed to answer. This list will vary from case to case but it will always include the elementary things like "Is each separate branch logically correct?" "Are there proper error returns for inadmissible data?" etc.

(2) Make the tests as noise-free as possible, but recognize that a large sampling is often required before a confident judgement can be formed. For instance, when testing to see whether a quadrature program reacts correctly to a small jump-discontinuity, it is largely a matter of chance that determines whether a function-evaluation is called for at a point which will make the discontinuity detectable. A program may detect a small discontinuity in one place and fail to detect a far larger one situated somewhere else. Give the program a hundred different discontinuity-detection problems of approximately the same difficulty and determine the percentage score.

(3) See whether the program shows any undue sensitivity to the mode of problem-formulation. Linear equation solvers are commonly quite sensitive to the scaling, and it is unnecessary to produce grossly mis-scaled examples to demonstrate this. Some are sensitive to the order in which the equations are written. A polynomial interpolator may work fine on three points situated at -1, 0, 1 and break down on the same three ordinates situated at 99, 100, 101. In summary, check that the program will take the necessary steps (e.g., scaling or origin-shifting) to minimize the numerical problems. Failing that, check that the documentation carries an appropriate "...may be hazardous..." warning.

(4) Wherever possible, relate the observed error(s) to some accepted standard or norm. The statement that "observed errors never exceeded .1% of the bound given by Wilkinson" is far more informative than "the errors never exceeded 1 in the fourth decimal figure."

(5) In presenting test results try to give the reader a maximum of information for a minimum of numbers. The performance of a linear equation solver can be well displayed in the form of a 4 x 6 array, where the row index defines the order and the column index defines the condition number. If each element of the 4 x 6 array is the average of 20 experiments, the array should have enough smoothness to permit "eyeball" interpolation. Any lack of smoothness suggests that further probing is called for.

(6) Remember that computer programs will never be used in earnest on problems for which the right answers are known analytically. It is unnecessary, and in some cases unwise, to restrict the tests to problems of this type. For instance, it is quite common for a quadrature program to tell us that:

$$\int_a^b f(x)dx \neq - \int_b^a f(x)dx \neq 0.$$

Without knowing the true value of the integral, we can say that one of the quadratures was wrong by at least half the difference. A minor discrepancy would be considered normal, but a major discrepancy in any respect (e.g., number or distribution of sample points) will indicate non-optimal programming.

(7) When an algorithm consists of several distinct parts, e.g., similarity transformation of a matrix, followed by eigenvalue determination, it is generally the best policy to test each part separately; however, there are exceptions, such as predictor-corrector pairs of formulas for solving ODE's which should be judged as a tandem.

(8) Remember that validation involves a determination of the limitations as well as the capabilities of a program. Parametric test problems should be used, and the parameter(s) varied so that the test problems become unsolvable in the given wordlength. Check to see how realiably a program identifies this situation.

A.C.R. Newberry
University of Kentucky

August 1972

ATTACHMENT

Testing Nonlinear Equation Solvers

Define a two-parameter family of polynomials by

$$P(x,p,k) = (x-p)^k (x-1)(x-2)(x-3)$$

and construct its coefficients A_r which are functions of p,k

p will vary in [0,1] and k will vary over 1,2,3.

The test consists in attempting to find the roots of P simultaneously by nonlinear equation solving thus:

(1) Make a (bad) guess $\bar{g}$ at the roots, e.g., for

$$k=2, p=1/2 \quad \bar{g}=0,1/4,3/4,4,5$$

(2) Construct the monic whose roots are the components of $\bar{g}$. Coefficients of this monic are b_r. Then a-b serves as a residual vector. Adjust $\bar{g}$ with a view to making $\bar{a}-\bar{b}=0$. The residual vector $h=\bar{a}-\bar{b}$ is a nonlinear mapping of $\bar{g}$.

(3) On concluding, the elements of $\bar{g}$ may have to be sorted monotonically so as to provide a best match with the known roots of P. After sorting define E as the largest absolute difference between an element of $\bar{g}$ and the root it approximates. If k, p are chosen so that P has a repeated root, the Jacobian will be singular, so this will be a severe test.

(4) Let k vary over 1,2,3 and p range over [0,1] in intervals of .25.

(5) For each k,p get an average E (see (3)). The average is taken over 5 starting vectors $\bar{g}$ for that particular problem. The starting vectors are uniform floating numbers in (0,5). Set average E negative if any failure occurs in the set of 5 tests.

(6) Output a 3x5 matrix of average-E values as defined above.

A.C.R. Newberry (1968)

QUESTIONS FOR SUBGROUP DISCUSSION OF TOPIC I

1. The term "certification" applied to software seems to have various meanings in different contexts; e.g., "certify" (run) an algorithm (CACM), certified software as supported software (NATS,IMSL). Is there an underlying concept which unifies these meanings or are several concepts being confused?

2. Is the demand for certification actually a demand for highly informative documentation about the performance of a code?

3. Can standards of computer program performance be determined and published through some form of structured committee action? What standardization process would you like to see evolve?

SUBGROUP DISCUSSION OF TOPIC I

Lloyd Fosdick -- I will try to capture the ideas as they were expressed. One point was that certification should consist of a precise statement of procedures used to test software. The statement should be sufficiently accurate to permit repetition of the tests and should include requirements for clarity and intelligibility in program documentation. The documentation for a program should be layered with the layers involving different levels of detail.

The group felt that evaluation should be distinguished from certification. The former involves a comparison of relative merit of the software with respect to other software, real or imagined, for the same purpose; hence it depends on the problem area and on the system. The latter, certification, focuses attention on one program, in isolation.

Those statements address our answers in one way or another to questions 1 and 2. We didn't really complete 3. The comments relative to that question are that the alliance should play a formal role in setting standards of performance. For instance, this would include such things as what should be in a performance profile. The alliance should collect standard test cases and should provide a place where software could be sent for testing purposes. Anybody else that was in the group have anything to add, especially regarding question 3?

Fred Krogh -- I see some problem in getting together and agreeing on what things are important in evaluating. I think it is a problem but

I do not have any suggestions on what to do about it. I think there is a tendency in evaluating software to evaluate things that are easy to evaluate and ignore things which are quite important but which are hard to evaluate.

Wayne Cowell -- Well, it is a little difficult to take a freewheeling discussion and summarize it in a very neat fashion, but I was struck as I heard Lloyd just now that there were several points that were common to the two groups. It might mean that they are very important points.

Let me start by stating the conclusion which we finally drew, which was that "certification" is meaningless unless much more is added to the term. In particular, you have to say who is going to do what. Although the term has been used a great deal, we realized very strongly in our discussion that it had been used very loosely and that there is a semantic confusion.

We distinguished among three aspects of certification: (1) validation of documentation and specification, (2) evaluation of quality, (3) support and maintenance of routines. One part of the meaning of certification is the question of placing a kind of stamp of approval or agreement by the certifier that the documentation is correct. We distinguished between validation in that sense and evaluation, which deals with quality and usefulness. It is possible for a subroutine to be correct and yet be irrelevant to the needs of a given group.

Another aspect of certification which we brought out was the idea of support, the fact that some group needs to agree to stand behind what was marked certified.

Having made the basic distinction between validation that the documentation is correct and the evaluation that the program is useful, we can then say some other things about these aspects of certification. For example, in terms of validating software, we could define a spectrum, a lower bound and upper bound. The lower bound on validation would be the fact that somebody had run it. This is what CACM does. The upper bound would be a proof of correctness. This would be the ultimate kind of validation. Now, even if we can prove a program correct and therefore have ultimate validation, that does not necessarily mean it is useful.

It was brought out that certification, no matter what meaning we associate with it, is a dynamic process. It goes on and has to change as systems change, since it is indeed relative to a given system.

There was some discussion in our group as to who does validation or evaluation? I believe that there was a concensus that ultimately there is a need for an independent certifying agency, that is, independent of the producers of the software. Both the evaluation of a routine and the validation of its documentation should be done by this independent agency. It was admitted that this situation does not exist yet today. Indeed people like the NATS group are, in a sense, certifying their own work and are, hopefully, defining building blocks that can be used later for the construction of these independent certification activities. You notice we did not follow the questions in the agenda explicitly, but I am satisfied that we covered their intent. Would anyone like to add any comments?

Jim Boyle -- The support question is an interesting one. In our group several people expressed opinions that it had nothing to do with certification. I think the role of the support in the NATS project was more of a way of lending credibility to the fact that we had done exhaustive testing. Saying we would support the routines was partly intended to convince people that we had tested it enough so the user is not letting himself in for trouble. I wonder if, in fact, we might see a diminuation in importance of the support as people come to believe more in the concepts.

DRAFT CERTIFICATION STATEMENT

Wayne Cowell - Acting on a suggestion from Joe Traub and others this morning, Lloyd Fosdick and I sat down this afternoon and prepared a statement about certification. Following is a draft for your examination and comment:

> Recognizing the need to be able to identify those computer programs which reach some accepted level of quality the term "certification" is useful. Ultimately a certified program is one which has been widely accepted within the communities of experts and users. To assure this credibility the process of certification must include examination of
>
> 1) completeness of program documentation,
> 2) performance of the program relative to its documentation,
> 3) comparison of the program with others of the same type in terms appropriate to the problem,
> 4) adequacy of continuing maintenance and support.
>
> A formal guarantee that the certification process has been satisfactorily performed would be expressed by a document issued by an agency or institution recognized by the communites of users and experts.

DISCUSSION TOPIC II -- EDUCATION AND INTERNSHIP IN SOFTWARE EVALUATION

DESIGN OF A COURSE ON ALGORITHM TESTING

A. C. R. *Newbery* - I will tell you first the way the course looks according to the catalog (CN-1). Something like 60% will be error analysis and the rest will involve writing programs to implement the error analysis. As I mentioned this morning, I attach a good deal of importance to relating observed errors to error bounds and seeing how realistic the bounds are. Also, this helps you to tell whether an observed error should be judged as being a good performance or a poor performance. There are two numerical analysis pre-requisite courses, one of them at senior level and one at junior graduate level. The senior level one deals with elementary error analysis, floating point arithmetic, failure of distributive rules, dangers of subtracting nearly equal large numbers. After that there is a 500-level course that is more conventional. With that background (and hopefully a good deal more) they go into this course on algorithm testing.

I think one could consider separately the first order approximation and the more rigorous norm-based Wilkinson type error bounds for inverses of matrices. On the eigenproblem, one would consider the very nice theories that are applicable to symmetric problems, and one would deplore the very unsatisfactory state of error analysis with respect to nonsymmetric problems. Error propagation in ODE's would be included. I am sure that is very familiar stuff which you would find in most courses. The PDE's are perhaps less frequently dealt with and you must consider severe constraints on your mesh size.

We would spend some time just looking at available test matrices--there are several books and publications containing useful test matrices, useful either for inversion or for checking of eigenvalue routines.

The last section would be the design and coding of programs to test for the errors and possible deficiencies in a program. One would try to implement this in dozens of different areas, but, rather than diversify too much, I have made a tentative list (not intended to be comprehensive) of questions which one might wish to ask with respect to polynomials. Given the routine, what questions do you ask and what questions do you require an answer to before you sign your name to it as being a good polynomial root finder? The student would have to figure out a decent way of finding answers to the following questions: Does the routine behave well with respect to the kind of polynomials which traditionally cause problems? Does it spin its wheels when it has found a root? i.e., does it keep iterating after it has found an acceptable sequence of zeros? Are they found in increasing order of size? Is the behavior consistent? Is the accuracy as good as the wordlength permits? The latter is probably the most important question and the least easy to answer in general although in the case of polynomials it is fairly easy to answer on the basis of a posteriori error analysis. I would expect students to solve that kind of problem with respect to polynomials as well as eigenvalues, ODE's and what not.

That provides an indication of what I want to to in the course.

C.S.631 ERROR ANALYSIS AND VALIDATION

Effects of inexact data and/or inexact arithmetic on the accuracy of the computed solution to a problem.

Design of acceptance tests on the basis of which a computer program may be certified as meeting its specifications.

10% Review floating-point error propagation.

20% Study matrix perturbation theory with reference to inversion and eigenproblems.

15% Error propagation in recursive computations, especially ODE's, PDE's, Bessel functions.

15% Backward error analysis for matrices, polynomials, Fourier series.

10% Test matrices, polynomials, etc.

30% Design and implementation of test programs which will test for any weakness in a program and give it "grades" over a spectrum of problems.

ALGORITHM EDITOR'S EXPERIENCE WITH STUDENT ASSISTANTS

Lloyd D. Fosdick - I think that when George Forsythe was editor of the Algorithms section he used students quite a bit in helping in various phases of the job, and I think it is also true that Jack Herriot did. I have been following a similar pattern. I want to say some words about the kinds of things that we have been doing in Colorado in this direction (LF-1). I might mention that the students involved here are present: Jacob Wu, Jeff Wright, and Dorothy Lang who has just graduated and is going to work for Texas Instruments right after this meeting.

Let me try, first of all, to identify some of the kinds of activity that students have been engaged in. These range from fairly simple and routine jobs (almost clerical but not quite) to ones that are fairly challenging. There is checking for violations of syntax standards. There is the maintenance of the algorithm index, including a cumulative index published by CALGO, with listings covering about 11 years. The index now has about 820 entries, so it is getting fairly long and it is a nontrivial matter to up date it each year. We have all of that material on punched cards, and somebody has to go in and bring it up to date on Certifications, Remarks, and so on with respect to a particular algorithm. Of course, all the new algorithms which have been published in all the journals have to be added.

Recently we have started an activity in which some of the longer algorithms are recorded on magnetic tape. Somebody has to worry about copying those tapes and seeing that they are checked and distributed properly. Students have helped create program aids and have helped develop several programs to help us in the testing. With one exception we have not really used these programs extensively for algorithms that are being published, but that is our intention.

As far as the tape distribution is concerned, I counted them just before coming to the meeting, and we have distributed 80 tapes on algorithms so far. This has been going on since April, 1971. I am a little bit suspicious that in many instances the orders for the tapes have been generated by some fairly automatic process like a librarian from a laboratory who orders all such things. The tape may be collecting dust somewhere. So I do not think that the number 80 is in any way a reflectin of how many of these have actually been utilized. The number of algorithms on a tape varies and is associated with a particular issue of the journal; usually it is one or two.

Let me say a word about whether or not these are good activities or bad activities as far as an educational process is concerned. I think that going into the literature and being aware of what is available and what is going on, is an educational process for a student. I do not think it is a good idea to have the same person doing the same thing for years and years on end. It is a good experience for a year to be involved with something like this. The creation of the tapes is interesting, and is a lot more than merely copying those tapes. One of the things that is involved is how much it is costing us to produce those tapes, and what should we be charging realistically in order to recover those costs. These are very practical matters of pricing that are important for the real world, and I think it is useful for students to be aware of them. Very often in graduate education, students go through a program and have very little idea of the monetary aspects of the things that they are doing. So for these reasons

I think there is some educational values in these activities, although they seem fairly routine.

Concerning the check for violation of syntax standards, I think one could reasonably ask why there is not a program we can use for checking syntax violations. The answer is there is a program and we use it. But these programs are not perfect and I will given you an interesting example of the sort of thing that is very difficult for them to handle. The problem is that the syntax is not perfectly specified; there are some ambiguities in it. According to ANSI standards a control variable in a DO-loop is undefined after exit from the loop provided you satisfy the DO conditions. So for instance (LF-2) if you went around this loop from J to K, and then dropped through from statement 30 to statement 40 and tried to execute the DO-loop, you would be violating ANSI Fortran. On the other hand, if you exit this top loop by means of the IF statement, it would not be a violation of ANSI Fortran. Now this is a complicated thing because it may well be that the person who constructed that particular loop may have decided that the loop never was satisfied, that you always did, in fact, exit through the IF statement. In that case one could not say it was a violation. Problems of this sort which are unclear are difficult to detect even by compilers which claim to have syntax checkers in them. There are situations of this sort, which our programs do not check. A student, usually Jacob Wu, goes over the programs after they are run through our so-called ANSI syntax checker. He makes a final check to see whether or not there are things that have been missed.

We have a program written by Dorothy Lang which takes Fortran programs and puts them in a stylized format with respect to spacing conventions and things like that. There are other programs around that do this sort of thing, but I believe this one is more sophisticated than the ones that are generally available.

I would like to say a couple of words about two other programs which we have developed. A branch analysis program is designed to help in checking programs, and there are some interesting stories connected with this which I would like to mention later. The program is supposed to recognize and isolate in a Fortran program what is called a basic block which is a sequence of consecutive statements in the source code which must be executed consecutively and therefore only the last statement can alter the normal flow of control. There cannot be any branches in a basic block except for the last statement. You can only get into it at the top and you can only leave it at the bottom. So only the first statement may be the terminal statement of the branch in the flow of control. We have a branch analysis program that takes as input a standard ANSI Fortran program and isolates basic blocks in the sense illustrated in this example (LF-3). This is a portion of a subroutine that was published in the CACM. The branch analysis program reads the original Fortran shown at the top of the slide and produces as output a new deck of cards shown at the bottom which would have the modifications indicated by the arrows. One of these calls to a name of a program, is inserted at the head of each basic block, the idea being to enable a subroutine call each time a basic block is entered. This permits analysis later when the program is actually undergoing execution. The first parameter in the call identifies the number of the basic block. The second parameter (it is 2 in every case here) can take on one of 3 different possible values, the others referring to whether the block appears to terminate in a stop statement, and therefore is one which exits completely from the program, or whether the block involves the first executable statement in the entire program.

If, as a result of executing each one of these calls to XNAMEX, we set a flag for that particular basic block, then we could see if flags for all the basic blocks were set in a series of tests. If they were, we could say that every statement in the program had been executed at least once. What you make this subroutine (called XNAMEX here) do is of course up to you. We have a program that merely set flags, and program that count numbers of executions (i.e., numbers of times we have been in that basic block).

Because of the structure of Fortran, things do not always break up easily into basic blocks. One awkwardness is in the logical IF. The way we handle that situation is a trick that was used in a program called FETE which was written at Stanford. In order to mark the fact that you took the exit from the block due to a true condition on a logical IF, we just duplicate the IF statement, and insert the call to the subroutine and repeat the IF statement again. If the test produces side effects, there is a problem. Normally that does not happen.

Another awkward situation is illustrated in this example (LF-4): The IF provides a branch outside the block headed by the DO. But one of the branches goes to the terminating statement in the DO loop. The only simple way to solve that problem is to first run the code through our STYLE program which will change this so all DO loops end on a CONTINUE statement and the situation can be handled. These little technical details in Fortran cause certain complications but basically we do have a tool that will allow us to recognize the control sequence followed during the course of execution of the program.

Now, what we then tried to do this summer was to look carefully at several algorithms to see how these programs might be useful for purposes of testing. For this purpose we took four algorithms which had been published in CACM (LF-5).

Before I get on to the results of that, let me make an interesting observation. I do not feel that the branch analysis program has been satisfactorally tested, because we have not completed running it on itself. We did all the insertions and verified that all the insertions had been correctly done, but we have not driven that program down to all of the basic blocks yet, so in that sense it is not completely tested.

But back to the four algorithms. I think I have observed some fundamental constant. These four algorithms involve an eignevalue-engenvector algorithm, a greatest common divisor algorithm, a quadrature algorithm, and Gear's algorithm for solving differential equations. The ratio of source statements (I am taking out the comments) to the number of basic blocks in the code is almost always a number that hovers around 1.8. When I first saw this, my initial reaction was that it seems to be extremely small because it says that you can only go about two statements before you go someplace else. There is that much jumping around. The fact that branch analysis is a non-numeric program and also is exactly in that pattern gives me some reason to believe that this probably is true for a large body of programs. This ratio always stands around 1.8. I don't know if there is any particular importance to that number, but it does interest me, because it is so low.

Now, this approach to looking at the testing situation is perhaps a little different from the kind of thing that has been discussed here so far, because here we are more interested in the structure of the program and are just verifying that the various logical paths have or have not been checked. We are not looking at proofs of convergence. In two of these algorithms we found errors as a result of going through this process. In one case we found code that could not possibly be executed. This was written by a careful person and looked at by two referees before it was published. The other algorithm that was looked at in which there were errors uncovered was a different situation. There it was easy to drive the program down all of the basic blocks. But in the process of doing that, erroneous answers were created. In trying to find data which forces execution down these basic blocks, one uncovered by serendipity data that produced wrong answers. There is an interesting side story. This particular algorithm was subject to a formal proof procedure and that proof uncovered exactly the same errors that we uncovered using this branch analysis program, although I think it took us a lot less sweat. On the other hand, we cannot be sure we have uncovered all the errors either.

The real goal of this particular line of work is quite ambitious. We have written and are testing a program called PATH which identifies the linkage between the basic blocks. With that information we can easily identify the possible paths in ourprogram. Then we would like to make a distinction between what I call syntactic paths and semantic paths. Syntactic paths are paths which superficially you might be able to execute but in fact you may never be able to execute. An illustration of that might be a branching process where at some point you go down a particular path when a particular variable is negative; but earlier you always computed the square of that variable so you know it is positive. The extent to which we can identify that sort of thing I do not know. It is easy to construct situations where it would be almost impossible to unravel the semantic paths but how often those situations occur in real program is not clear until we look at them. This is a line of future work.

STUDENT ACTIVITIES

- Check for violation of syntax standards.
- Maintain algorithm index.
- Create, test algorithms tapes for distribution.
  Monitor costs.
- Help create program aids. e.g., STYLE, BRNANL, PATH
- Use program aids for formulating and testing.

Slide LF-1

```
DO 30 I = J,K
 ≡
IF (...) GO TO 40
 ≡
30 CONTINUE
40 DO 50  L = J,I.
```

Slide LF-2

```
          SUBROUTINE GCDN(N,A,Z,IGCD)
          DIMENSION A(50),Z(50)
          INTEGER A,Z,C1,C2,Y1,Y2,Q
          DO 1 M = 1,N
          IF (A(M).NE.0) GO TO 3
     1    Z(M) = 0
          IGCD = 0
          RETURN
     3    IF (M.NE.N) GO TO 4
          IGCD = A(M)
          =
```

```
          SUBROUTINE GCDN(N,A,Z,IGCD)
          DIMENSION A(50),Z(50)
          INTEGER A,Z,C1,C2,Y1,Y2,Q
 ---->    CALL XNAMEX (0001,2)
          DO 1 M = 1,N
 ---->    CALL XNAMEX (0002,2)
 ---->    IF (A(M).NE.0) CALL XNAMEX (0003,2)
          IF (A(M).NE.0) GO TO 3
 ---->    CALL XNAMEX (0004,2)
     1    Z(M) = 0
 ---->    CALL XNAMEX (0005,2)
          IGCD = 0
          RETURN
 ----> 3  CALL XNAMEX (0006,2)
          =
```

Slide LF-3

Example of awkward situation

```
      DO 20 J = 1,K
       X(J) = X(J) + Y
      IF(X(J))  10,20,30
10    L = L+1
20    V(J) = 0
30    A = B+C
```

Slide LF-4

- Algorithm 384 (Stewart)

  Eigenvalues and eigenvectors of a real symmetric matrix

- Algorithm 386 (Bradley)

  Greatest common divisor of n integers

- Algorithm 400 (Wallick)

  Modified Havie integration

- Algorithm 407 (Gear)

  DIFSUB for solution of ordinary differential equations

A	#SS	#BB	$\frac{\#SS}{\#BB}$
384	185	116	1.6
386	58	33	1.8
400	72	35	2.1
407	304	177	1.7
BRNANL	1125	605	1.8

QUESTIONS FOR SUBGROUP DISCUSSION OF TOPIC II

1. How may a computer science department include software evauation in the curriculum, either as a distinct course or contained in other courses?

2. Can student assistants participate in testing software so that a) useful tests are performed leading to acceptability decisions, and b) the student has an educational experience recognized by the university?

REPORT OF SUBGROUP DISCUSSION OF TOPIC II

Henry Thacher and David Young (reported by HT)

Henry Thacher - This is a joint report which has an advantage for a reporter. When you report on two groups at once, the audience thinks that any of your personal prejudices which you introduce were suggested by the other group, and therefore I have a free hand! Furthermore, I have rearranged some of the discussion so as to make it a little more coherent.

In considering the first question we asked what are we educating people for? There seemed to be three forms of education which got attention, or at least were mentioned. In the first place there is the man who is going to be the specialist in certification and mathematical software. We are not at all sure how many of these are needed or how many people we can attract into the area. The second level of education is education of the general user such as the engineer who takes service courses and general computer scientists. We would hope to persuade these people that testing software is a worthwhile activity and that they should demand that the packages that they get from somewhere else have some assurance of quality. Finally, we recognized a need for specific education for the users of systematized packages. This includes library operation and customer relations for the computer center but we should remember that many of the commercial people who market software also market quite extensive seminar training programs on the use of their software packages.

There seemed to be fairly general agreement from both groups that, for a variety of reasons, special courses in certification and testing may not be advisable at the moment. It comes down to questions of how many people you need to train as specialists. Such a course generally turns out to be rather specialized. It was pointed out that certification and testing can be a successful basis for seminars and small group project-type work. There was also a consensus in our group (I gather there were some reservations in the other group) that software evaluation should be emphasized in all courses, starting with elementary programming. In these courses you would be talking about good programming practice, which is a more general topic than certified numerical software. And certainly, when you get to the introductory numerical methods course, whether it is a service course or one restricted to computer science and mathematics students, the topic of evaluation testing and certification is extremely important to raise. There are some texts now that include a consideration of quality.

The question on student assistant participation got transformed slightly because it was obvious that the question has been answered in the affirmative by the presentations. But it did open up discussion of a form of education which may be appropriate for this work; namely, apprenticeship as a training or educational experience. I think that this was rather highly favored in both groups. This material does not lend itself to enthralling lectures. It is much easier to motivate a student to the care that is required by actually having him stub his toe over some of these problems in a real-life situation. There was a caution that such apprenticeship programs should not simply be another means of support for students, but they should be, as much as possible, reserved for students who are fairly well-motivated in this area and hope to continue in it. It is frustrating to put the effort into training an apprentice who is not really interested.

As soon as he gets through, he leaves the area without having contributed much for the money and effort spent on him.

You might build such a program in the computer center. It depends on whether the local center has staff to supervise the students, because they are not going to learn much of they do not get good supervision. Other examples, Argonne has some co-op programs and the Bureau of Standards also has summer student programs. The presidential internships at the Bureau have been very stimulating programs for students who have been privileged to participate. The problems in the university are the kind which can probably be licked. The major problem is how you can give academic credit for this type of experience.

I guess that covers our summary.

DISCUSSION TOPIC III - RESEARCH ON TESTING, PORTABILITY, AND LIBRARY DEVELOPMENT

PORTABILITY PROBLEMS AND SOLUTIONS IN NATS

James Boyle - I think the title of my presentation, "Portability Problems and Solutions" is perhaps more one of convenience than content. But I hope what I have to say bears on testing, portability, and library development to some extent. It also ties in with what Lloyd said this morning, but in the NATS project we were faced with a somewhat different set of problems than he has faced in the algorithm section of CACM, and so we have attacked different facets of the portability problem. This whole question of calling subroutines "portable" always makes me a little uneasy and my experience in NATS has confirmed that. It reminds me a bit of the situation around 1960 when several manufacturers were advertising what they called "portable" high fidelity sereo systems. Actually, they were only portable if you were Siamese twins or a member of the Green Bay Packers. I think they coined the word "portative" for something that was almost too heavy to carry and it seems to me that's the way subroutines are--at best, they're portative. I don't see this situation changing soon.

What I say is based on my experience in the NATS project and, more specifically, with the EISPACK eigensystem codes. Wayne reviewed the project for you this morning but I want to underscore a few points which bear on what I have to say. Basically the project consisted of taking software which was in good shape, making it available to a group of test sites for testing on specific machines, implementing their suggestions for improvements and correcting their reported bugs, making the package available for re-testing and finally sending out the so-called "certified" version after it had successfully completed testing. We felt early on in the project (partly influenced by Brian Ford of NAG) that we needed to maintain very good central control over the distribution of this material. In other words, we wanted to be sure that the material which was tested was the same as that which was finally distributed as certified, in so far as this was possible. We did all we could to discourage the test sites from making any changes to routines if they found anything wrong. Rather we encouraged them to give the information back to us (perhaps after they had tested the proposed correction themselves). We then made changes in our master versions and sent the routine out again in its corrected form for re-testing.

In regard to the portability question, the special function codes in the NATS project are specifically nonportable--they are tailored for each machine. On the other hand, we endeavored to make the EISPACK matrix codes as machine-independent as possible. Most of the machine dependency is contained in a couple of constants which reflect differences in computer arithmetic. Aside from these we have basically only two versions - one in REAL*8 for double precision arithmetic on IBM 360 and a single precision version which we very optimistically called ANSI Fortran. But even having minimized the numerical differences between routines, we discovered (or rather we had reaffirmed) that there are many, many differences between compilers, even between two different compilers for Fortran on the same machine. In fact, we couldn't get by with just one ANSI version.

We saw one example of such a difference when we decided we wanted to declare all the identifiers appearing in the routines, including the names of Fortran intrinsic functions. According to the ANSI standards it is perfectly legal to declare those names (and WATFOR required such declarations for double precision), but a couple of compilers balked at the declarations, one of them complaining that it was illegal and one believing that we intended to supply our own square root routine.

So despite our initial desire to have only two versions, we did, in fact, wind up with six. It reaffirmed for us this whole lesson that there is no such thing as one portable subroutine. You've got to have many versions. So the problem we faced was how to cope with essentially six versions of about 35 subroutines together with six versions of some 12 test drivers.

We wanted to get the machine to take care of the routine work by developing special purpose information processing programs. "Special purpose" means that the programs take advantage of properties of the package of routines that we are working with. We wanted programs to assist us in keeping track of the various versions. Our intent was to produce a condensed or composite storage of the routines enabling us to maintain one file rather than six. This file would contain the card images corresponding to the subroutines together with control information about which machine versions the particular cards belonged to, subject, of course, to the constraint that a card which appeared in more than one version would appear only once in this master file. The obvious advantage is that if the versions are similar the total amount of composite storage space will be less than would be required for the individual versions. But even more significant, as far as we were concerned, is the fact that we had only one file to take care of. That meant that, if a bug were uncovered in testing or some suggested change were to be made, it needed to be done in only one place. This structure for the file permitted us to see whether a change affected only one version, or versions for several machines. It gave considerable assurance that necessary changes were made in all the relevant versions.

We used a program to derive each specific machine version from the master file when we wanted to send it out to the test sites and as long as that program was not changed, we could repeat the derivation when the routines were certified and ready to be publically distributed.

Here (JB-1) is a segment of one of the routines as it appears in the master file version. The control cards are marked with the asterisk in column 1; they indicate which versions the following cards belong to. Here (JB-2) is the IBM 360 code derived from this. Note that the constant MACHEP has been set in the derivation process and serialization provided.

For the EISPACK codes we could characterize relatively simply the differences between the 360 double precision versions and the ANSI version and so we did a program which produced the master file version from the 360 double precision version. It was very special purpose, but it did provide us with further flexibility. I think that there is some further work to be done in this area. For example, one thing we don't have at present and which would be very useful in the context of a software alliance, is a way of combining an adaptation of a particular program for some machine with a master file version existing for other machines. Clearly that's a non-simple problem.

Having completed the development of these programs, we began to confront the question of what we were going to do with our documentation. When the routines were orginally sent to the test sites they contained a number of specific references to Argonne, and to IBM 360 long precision arithmetic, etc. We wanted to produce what we call NATS documents - that is, documents that apply to all of the versions of the code. We felt we could make the documentation portable; i.e., make one set of documentation that applies to all machines with the appropriate references or description of the variations between machines. It was natural to make these modifications by means of information processing programs since we had been fairly careful in preparing the documentation that it all adhere to a uniform format; in fact many of the documents were prepared from one another using our text-editing system.

The changes suggested for the NATS documentation fell into two classes: format changes (changes to be made uniformly throughout all the documents) and corrections resulting from revisions or bug-detection. When making corrections you have to confront the question of line justification and what happens if you need to insert a word at the beginning of a paragraph. We wanted our solution to do something about that. We did not have any text editing facilities available that incorporated even a rough, or ragged, right margin justification but we had developed a program to take care of that given certain "tabulation information" in the beginning of each line. We decided to make the document processing programs insert the tablulation information so we could use the line justifier.

I have a slide here (JB-3) showing a little of the history of the processing of these documents. We began with the AMD subroutine library document which was in all capital letters, had been prepared in machine readable form, was specific to the IBM 360 and to AMD, and we applied what we call a standardizer program that cleaned up the format and inserted some blank lines. Then the tabber program inserted the information that would be used for ragged right margin justification. This was done quite heuristically; the tabber had to make some guesses about what was intended in the format of the document and so the documents required some, although not a great deal, of hand adjustment to make sure that the justification process would work correctly. At this point, then, we had the documents in a form where corrections could be made readily by hand. If a word needed to be inserted we could just insert it between lines and adjust only the adjacent lines. When that was done we then had the possibility of applying the justifier and the pager program to get a correct, justified, AMD document for our purposes. In addition we could apply a converter program that made the format changes and additions for the NATS document.

After the NATS documents had been completed, we then began to consider the question of publication of a user's guide. We wanted to have documents for this in upper and lower case, properly capitalized, and also to edit out the sections that had been inserted in the NATS documents which were essentially common so they could be collected in one place. We did an adaptation of the converter to produce this publication form. Then we did a capitalizing program for the documents. This was written by Bruce Chapman, a summer student from the University of California; he has just completed this program under my direction. By taking advantage of the format of the documentation and of tabbing information that had been inserted he was able to write a program that did the capitalization completely correctly without any hand intervention. We will print these justified pages on a magnetic card selectric typewriter (MCST) functioning as a computer terminal and then photographically reproduce them for publication.

Of course the construction of all these programs raises the question of whether it was any less work to do the programs than it would have been to do the stuff by hand. It was certainly more sanity-preserving! And I think in this case it was actually more efficient because we didn't anticipate doing the publication version of the documents when we began. Except for the capitalization effort, we got it essentially free. Certainly producing the information processing programs was more fun than changing the documents by hand, and it also avoided the question of reproofing them.

I just want to add that production of these programs was facilitated to some extent by the use of the properties of the package and also by modular design of the programs--that is having sections that were used in more than one program. I think that we need further study of how one structures a program to make it easy to produce modifications.

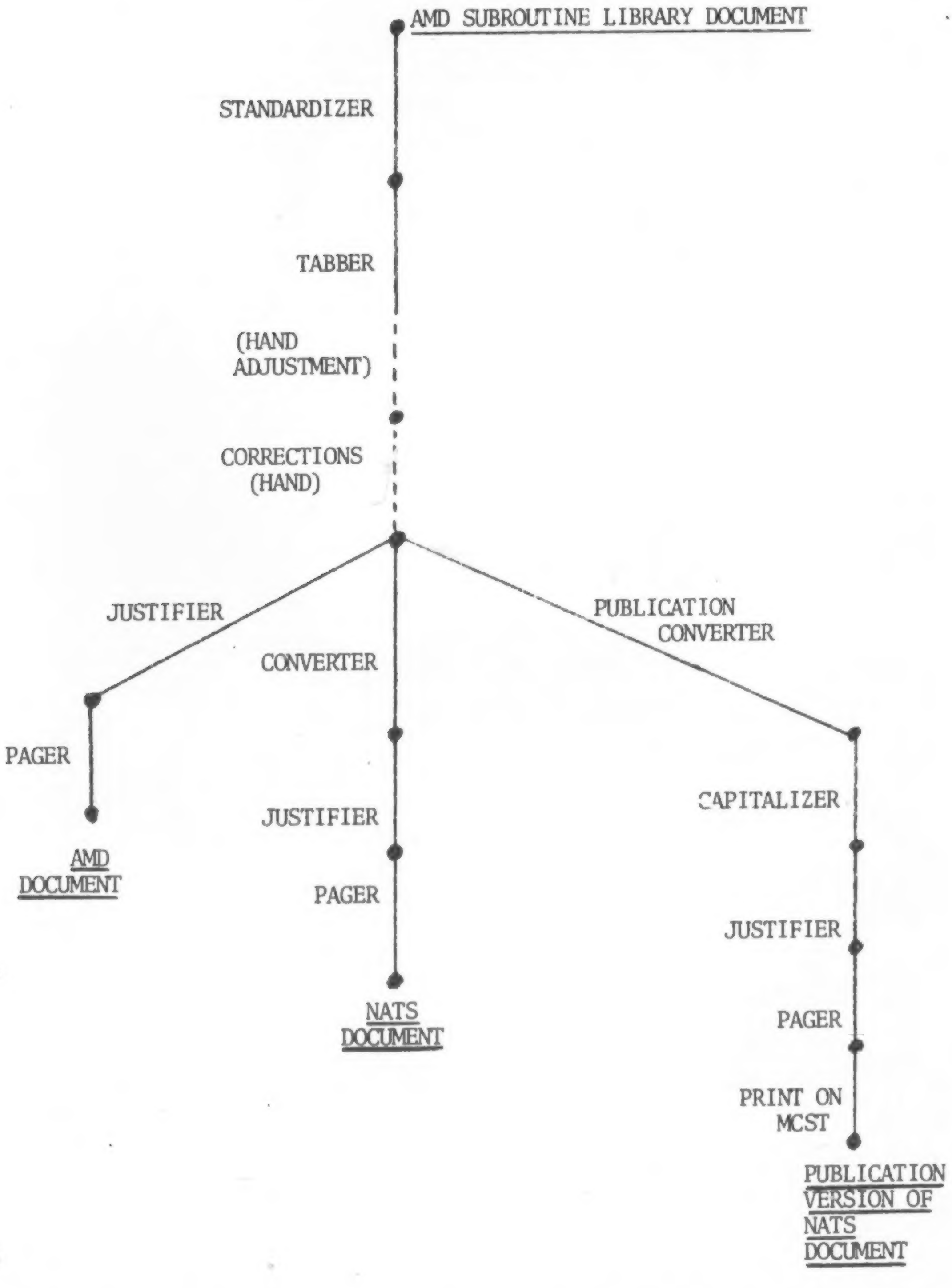
To summarize I've alluded to two main points about any effort like NATS for testing programs. The first is in regard to portability and states that, standards notwithstanding, you will probably have to have a version for each machine, and you should plan for that from the beginning. The second is that testing of packages of related codes is probably much more economical than testing an equivalent number of unrelated codes because it becomes possible to do programs of this sort to assist in the routine aspects.

```
C     ********** MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
* ALL
C                THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.
* 360D
C                MACHEP = 16.0D0**(-13) FOR LONG FORM ARITHMETIC
C                ON S360 ::::::::::
      DATA MACHEP/Z3410000000000000/
* ANSI
C
C                **********
      MACHEP = ?
* ALL
C
      IERR = 0
      IF (N .EQ. 1) GO TO 1001
C
      DO 100 I = 2, N
  100 E(I-1) = E(I)
C
* 360D
      F = 0.0D0
      B = 0.0D0
      E(N) = 0.0D0
* ANSI
      F = 0.0
      B = 0.0
      E(N) = 0.0
* ALL
C
      DO 290 L = 1, N
         J = 0
* 360D
         H = MACHEP * (DABS(D(L)) + DABS(E(L)))
* ANSI
         H = MACHEP * (ABS(D(L)) + ABS(E(L)))
* ALL
         IF (B .LT. H) B = H
* 360D
C     :::::::::: LOOK FOR SMALL SUB-DIAGONAL ELEMENT ::::::::::
* ANSI
```

Slide JB-1

```
C     :::::::::: MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING     89220047
C                THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.   89220048
C                MACHEP = 16.0D0**(-13) FOR LONG FORM ARITHMETIC        89220049
C                ON S360 ::::::::::                                      89220050
      DATA MACHEP/Z3410000000000000/                                    89220051
C                                                                       89220052
      IERR = 0                                                          89220053
      IF (N .EQ. 1) GO TO 1001                                          89220054
C                                                                       89220055
      DO 100 I = 2, N                                                   89220056
  100 E(I-1) = E(I)                                                     89220057
C                                                                       89220058
      F = 0.0D0                                                         89220059
      B = 0.0D0                                                         89220060
      E(N) = 0.0D0                                                      89220061
C                                                                       89220062
      DO 290 L = 1, N                                                   89220063
         J = 0                                                          89220064
         H = MACHEP * (DABS(D(L)) + DABS(E(L)))                         89220065
         IF (B .LT. H) B = H                                            89220066
C     :::::::::: LOOK FOR SMALL SUB-DIAGONAL ELEMENT ::::::::::         89220067
```

Slide JB-2


AMD SUBROUTINE LIBRARY DOCUMENT
STANDARDIZER
TABBER
(HAND ADJUSTMENT)
CORRECTIONS (HAND)
JUSTIFIER
CONVERTER
PUBLICATION CONVERTER
PAGER
AMD DOCUMENT
JUSTIFIER
PAGER
NATS DOCUMENT
CAPITALIZER
JUSTIFIER
PAGER
PRINT ON MCST
PUBLICATION VERSION OF NATS DOCUMENT

Slide JB-3

UNIFIED STANDARDS APPROACH TO TESTING

Walter Sadowski - So far I've heard a lot of mention of quality and certification and so on but I have not heard anyone talking about certifying or standardizing on the testing tools. Before one can measure the length of anything, one has to develop some benchmarks that are acceptable to everyone else. Essentially this is what I am going to talk about.

We have been testing a library of Fortran elementary functions put out by a major manufacturer. What comes out is an interesting fact that although the library is a very high quality library, it has errors in almost all of its routines. The errors are not usually serious; error traces are not made in certain function values; very small arguments or overflows occur and are not signaled; errors of judgement take place so that the routine actually gives away some accuracy that it does not need to. I am mentioning these facts to point up one reason why library testing is not adequate and that is that testing by a single individual or single group usually would be inadequate to uncover all the mistakes. What I would like to propose is a way of bringing all of the testing community into the testing business where each group or each person can contribute his special know-how to expand the test or possibly refine it in some way. I am going to be using jargon which is used at the National Bureau of Standards for calibration.

What we do is choose a primary standard consisting of a tape containing a set of arguments and a set of function values intended for a particular machine. The reason each machine or each system, if you wish, has its own standard state is because we would like to test some specific features of hardware and software, number bases, and so forth. Moreover the primary standard will be based on considerations of the mathematical behavior of the function. In order to make the primary standard workable, we have what I call a transfer standard, which is an algorithm used to generate function values. The algorithm is usually written in ANSI Fortran and is as portable as one can make it. It's primary feature is that it is extremely simple logically. If possible, a power series will be used to grind out all the function values. Thus if anyone wants to look at this algorithm, it is easy to see what the algorithm is doing. Presumably this will give the user more confidence. It is a brute force approach and you have to pay for it in some way. The use of such an algorithm is based on the use of an extended precision package which essentially uses so many figures in the calculation that truncation errors become unimportant. Such a package written by Maximon exists at the National Bureau of Standards, and we have extensively used it.

The third standard, which is the working standard, is a set of arguments and function values which have been chosen for a specific algorithm. For example, suppose someone has written a sin outline. It has its crossover points and its regions of difficulty. One designs a set of arguments for this specific algorithm, uses the simple transfer standard to generate the function values and then uses a test package, such as the package we developed, to do a bit comparison of the function values supplied to the transfer algorithm and to the algorithm to be tested.

Now, how would one use such a procedure? Somebody gets a tape and an algorithm and either uses his own extended precision package or a package sent to him. He puts the tape on his computer, he compiles the transfer algorithm, and the extended precision package, if it is not his, and he generates function values from the arguments on the tape. Then an automatic package will compare the function values generated with the function values on the tape, and they had better be identical. This tells you that you have calibrated your transfer algorithm, that it has compiled correctly, and that it is working in a healthy environment. This transfer algorithm can then be used to generate another tape, which will test any other algorithm that is of interest. We have here a purely non-portable approach because we believe a specialized tool is the most efficient tool to use for any particular job. There is, however, a portable link, namely the transfer algorithm which is not tailored to any particular machine. By providing such a portable link the matter of portability and transportability may become less acute and fewer people will be tempted to legislate rules for other people.

As I said, this testing scheme only concerns the mathematical functions of one variable so far. When we go to functions with many variables the experience of an applied mathematician becomes extremely important when arguments are chosen. We are now testing a function of one variable with 6,000 to 10,000 arguments. When you include another variable, the number of possible arguments becomes astronomical and one must use good judgement as to which density of arguments to apply where. This is where the testing community, by using the tapes, could be very helpful in setting up a kind of consensus standard as a tape with arguments and function values.

SIGNUM 1971 NONACADEMIC NUMERICAL MATHEMATICS SURVEY

F. N. Fritsch - During July and August, 1971, a questionnaire was sent by SIGNUM to some 447 nonacademic institutions through North America. The purpose of this survey was to attempt to determine where numerical mathematicians are employed in the nonacademic world, how many there are, and what they are doing. The results of this survey were, in a certain sense, disappointing. I did not obtain as much information as I had hoped. Of the 447 institutions to whom the surveys were sent, a total of 86 responses were obtained, about 20%. Of these, I considered 83 to be usable responses. Unfortunately, several institutions known to employ numerical mathematicians did not respond. One item on the survey which I think is particularly appropriate to this conference was a list of mathematical certification projects. The question was, "Is your institution engaged in a mathematical subroutine certification project? If so, please name the responsible individual." These were the institutions that reponded "yes". (FF-1) JPL has one too, but the return was too late to be included in the survey.

Let me quickly go through other results of the survey, for what interest they may have. Slide FF-2 shows the distribution of responses by zip code area, where 10 is Canada and 11 is Mexico. Responses from the west coast and the Washington, D.C. area predominate, and the distribution in the central part of the country is pretty uniform. Other questions concerned the source of funding. Some 53% said "Business and Industry," 34% said "Federal Government." In regard to principal activity, 49% said "Research and Development," 22% said "Other", whatever that is. (Apparently, I made a bad guess at activities.)

More interesting was response to the question, "How many numerical mathematicians do you have employed at your institution?" I asked for a breakdown by degree level. It was claimed that 2,096 people were currently employed as mathematicians by these 83 institutions. Of those 1099 were bachelors level, 544 masters level, 361 doctorates, and 92 nondegreed. We asked where these numerical mathematicians were located in the institution; i.e., in special project areas, computing center, special numerical analysis area, etc. Some 60% of these were in special project areas, 10% were in computing centers, 15% in mathematics divisions, 4% in special numerical analysis sections, 10% somewhere else. I do not know where somewhere else could be. We found that more than half of the mathematicians were on the west cost (zip code 9) and that was the only zip code with a significant number of nondegree numerical mathematicians.

We took a look at the job titles used to classify numerical mathematicians. Nobody included "numerical mathematician" as a job title. There were such things as analyst, engineers of various types, programmers, laboratory scientists. In particular, I looked at the responses of one institution, which supported some 50% of the total for the zip code 9 area. They had titles such as "customer engineer," that sort of thing. Apparently anybody who had some mathematical background was considered to be a numerical mathematician. When I dropped that particular institution and a couple of these that had zeros all the way across, from the population, I got what I thought was a slightly more meaningful breakdown (FF-3).

I think that in order for any survey of this type to really have validity, we need to do a better job of defining our terms. What do we mean by a "numerical mathematician?" I offered several tentative definitions in the SIGNUM Newsletter [3] but have not had any feedback yet.

References:

1. "SIGNUM 1971 Non-academic Numerical Mathematics Survey, A Brief Summary," SIGNUM Newsletter (April 1972), pp. 14-18.

2. "SIGNUM 1971 Non-academic Numerical Mathematics Survey," Lawrence Livermore Laboratory Rept. MISC-00777 (April 14, 1972).

3. "Toward a Definition of the Term Numerical Mathematician," SIGNUM Newsletter (April 1972), p. 19.

LIST OF MATHEMATICAL SUBROUTINE CERTIFICATION PROJECTS

Fred Meyer
The Aerospace Corporation

W. J. Cody
Argonne National Laboratory

Denny D. Sutherland
The Babcock and Wilcox Company

Samuel L. S. Jacoby
Boeing Computer Services, Inc.

Andrew Schoene
Burroughs Corporation

W. C. Richie, Jr.
Computer Knowledge Corporation

David Uslan
Computer Sciences Corporation

Mansfield L. Clinnick
Lawrence Berkely Laboratory

Frederick N. Fritsch
Lawrence Livermore Laboratory

Hans J. Oser
National Bureau of Standards

M. G. Singleton
North American Rockwell Corporation

Paul Oliver
Sperry Rand - UNIVAC

G. W. Westley
Computing Technology Center

Milton Reese
U. S. Naval Postgraduate School

Elizabeth Cuthill
U. S. Naval Ship Research &
Development Center

Charles Lawson
Jet Propulsion Laboratory

Slide FF-1

A. INSTITUTIONAL IDENTIFICATION

Table 1. Number of responding institutions, by U.S. postal zip area.

ZIP AREA	0	1	2	3	4	5	6	7	8	9	CANADA	MEXICO	TOTAL
NUMBER	6	7	16	4	4	5	4	4	4	24	5	0	83
PERCENTAGE	7.2	8.4	19.3	4.8	4.8	6.0	4.8	4.8	4.8	28.9	6.0	0.	

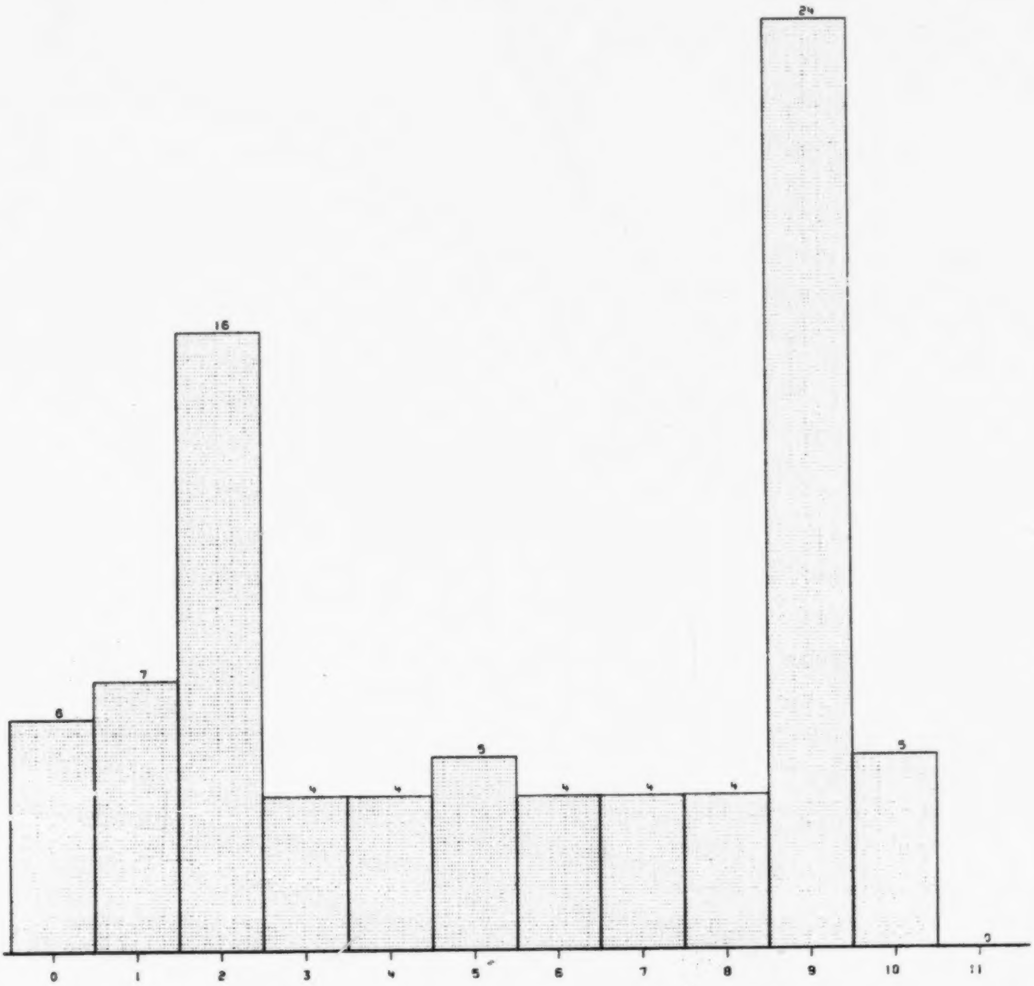


Fig. 1. Number of responding institutions by U.S. postal zip area.

Table 2. Sources of funding and principal activities of responding institutions (numbers in parentheses are percentages).

SOURCE OF FUNDING	
BUSINESS OR INDUSTRY.....	44 (53.0)
FEDERAL GOVERNMENT.......	28 (33.7)
STATE/LOCAL GOVERNMENT...	0 (0.)
NONPROFIT ORGANIZATION...	6 (7.2)
OTHER....................	3 (3.6)
MORE THAN ONE RESPONSE...	0 (0.)
NO RESPONSE..............	2 (2.4)

PRINCIPAL ACTIVITY	
RESEARCH AND DEVELOPMENT.	41 (49.4)
PRODUCTION...............	8 (9.6)
SALES AND MARKETING......	5 (6.0)
OTHER....................	18 (21.7)
MORE THAN ONE RESPONSE...	3 (3.6)
NO RESPONSE..............	8 (9.6)

Slide FF-2

Table 2'. Sources of funding and principal activities of responding institutions (numbers in parentheses are percentages).

SOURCE OF FUNDING	
BUSINESS OR INDUSTRY.....	41 (52.6)
FEDERAL GOVERNMENT........	28 (35.9)
STATE/LOCAL GOVERNMENT...	0 (0.)
NONPROFIT ORGANIZATION...	5 (6.4)
OTHER....................	3 (3.8)
MORE THAN ONE RESPONSE...	0 (0.)
NO RESPONSE..............	1 (1.3)

PRINCIPAL ACTIVITY	
RESEARCH AND DEVELOPMENT.	41 (52.6)
PRODUCTION...............	8 (10.3)
SALES AND MARKETING......	4 (5.1)
OTHER....................	14 (17.9)
MORE THAN ONE RESPONSE...	3 (3.8)
NO RESPONSE..............	8 (10.3)

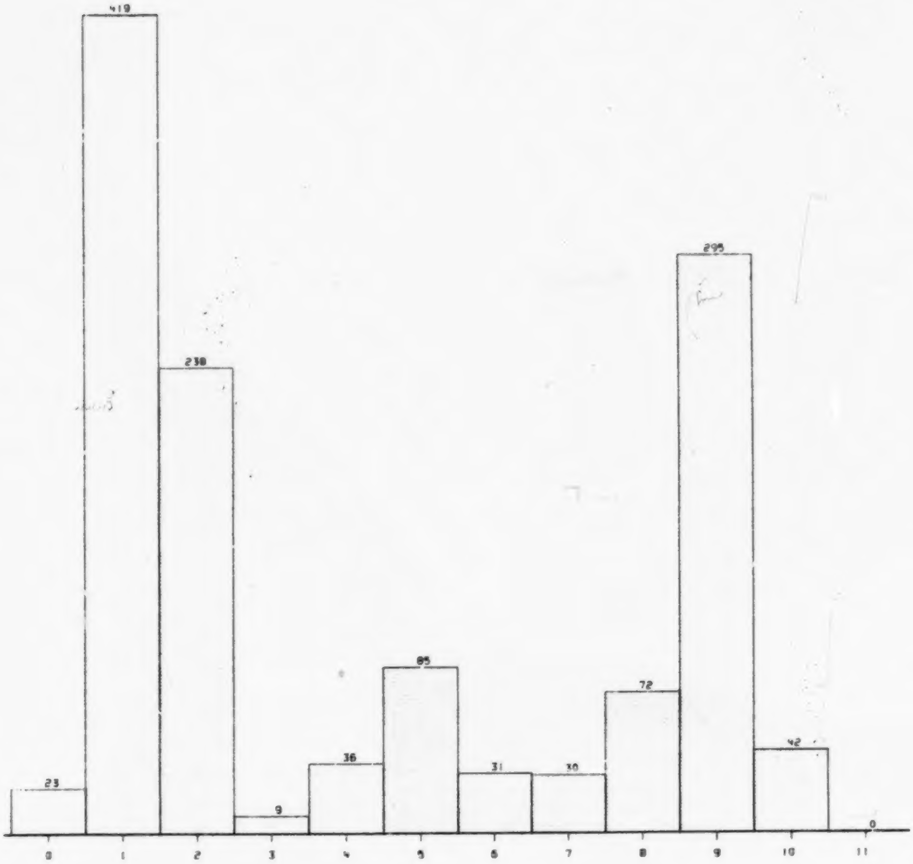


Fig. 2'. Current total number of numerical mathematicians, by U.S. postal zip area.

Table 3'. Number employeed by responding institutions.

	MINIMUM	MEDIAN	MEAN	MAXIMUM	TOTAL
NUMBER OF EMPLOYEES....	0	600.0	2708.2	50000	211240
NUMBER PROFESSIONAL....	0	245.0	671.9	10000	52407
PERCENT PROFESSIONAL....	0	50.0	44.6	99	

Slide FF-3

AN INTERACTIVE SYSTEM FOR STUDYING SEMANTIC MODELS OF COMPUTER PROGRAMS

Richard Fairley - I am interested in the development of software tools to aid in the testing of computer programs. My primary motivation in presenting this information to you is to get some feedback as to whether the tools I am proposing would be useful to people with your interests. The "summary of discussion" which was in your packet provides some motivation. Item II.A suggests that there should be a distinction between people developing and writing programs, and the activities concerned with field testing and certifications. Moreover, the summary claims that this leads to an important distinction between basic research in the area of testing and evaluation, and actual field testing of programs. I would place myself in the first category of basic research in testing and evaluation and feel that what I am doing fits with Item C of the Summary, where support was expressed for developing tools which would aid in testing and certification of software.

The system I am proposing is an interactive on-line system. It interfaces rather nicely with the work Lloyd Fosdick described earlier. I should also mention that this work is closely related to Bill Hetzel's work at the University of North Carolina, and to the EXDAMS system developed by Bob Balzer at Rand. The user at the computer terminal interacts with his program and views models of the execution of that program. The models display various attributes of the program which are of interest to the user.

The design methodology for this system is based on a pre-processor which processes the user's program to construct a data structure model of the syntax of the program, and to insert subroutine calls which will gather the history of execution when the program is run. The user does not interact with his executing program, but rather with the collected data. I hope to illustrate that there are some advantages to this approach.

Let me show you a block diagram of the system (RF-1). We see the source program and the input data. Processing results in producing abstract syntax and the execution history for the program. After the program has been executed, the user at the console communicates with model construction routines which are extracted from the library of models. The user has the capability of entering new sets of input data to generate a new execution history, or if corrections are needed in the source program, to modify the source program. He may then regenerate the abstract syntax and the execution history of the program.

I am not going to describe the data structures by which the syntax and the execution history are represented, but I will mention the set of primitive accessing functions. We are making an effort to isolate the data base from the tools that are constructed using these accessing functions, because we do not feel that the typical user of this system should be concerned with whether the execution history spills out onto tape or whether it is on disc, etc.

The main idea of the system is that the execution history is appropriately cross referenced to the abstract syntax enabling one to look at any point in execution. One can look at the values which have been computed up to that point, to determine how those values depend on other values that were computed, and to associate values with program text. We believe the primitive accessing functions will enable one to reconstruct exactly the execution history of the program.

Given the present state in the execution history, one wants to know what happened next. One might want to know just the next thing that happened -- whether it was a transfer of control, the calculation of a value and assignment, whatever. You might want to know something about the next variable name that was assigned a value, or the next or the previous type of transfer of control which occurred. You might want to know the value assigned to the next variable, or the next executed statement, or how did the flow of control go? One might qualify the inquiry by asking for the next relevant piece of data in some sense. It may be important to be able to set the position pointer in the execution history; e.g., at the start of execution, or at some particular statement on its first (or jth) execution.

Some types of program models which might be useful are: Structural models, flow of control, variable dependency, data sensitivity, and timing. One might want to see a structural overview of the subroutine relationships; or perhaps you wish to suppress the code in the source program and see only the structure of the IF statements. Flow of control models are self-explanatory. There are some very nice models of block structured processes which are well-adapted to this system. Some of you may be familiar with the contour model of block structured programs.

Variable dependency is self-explanatory; how was the value of a variable assigned in the program influenced by the assignments of previous values to other variables. Another technique which is possible, if you have the execution history in a data base, is flow-back analysis, in which you might present to the user a tree structure of the current statement of interest to show how the value assigned was influenced by previous values that were computed in the program. The ability to associate values and how they were determined with the program text which produced those values, is a very valuable tool on which higher level tools can be built.

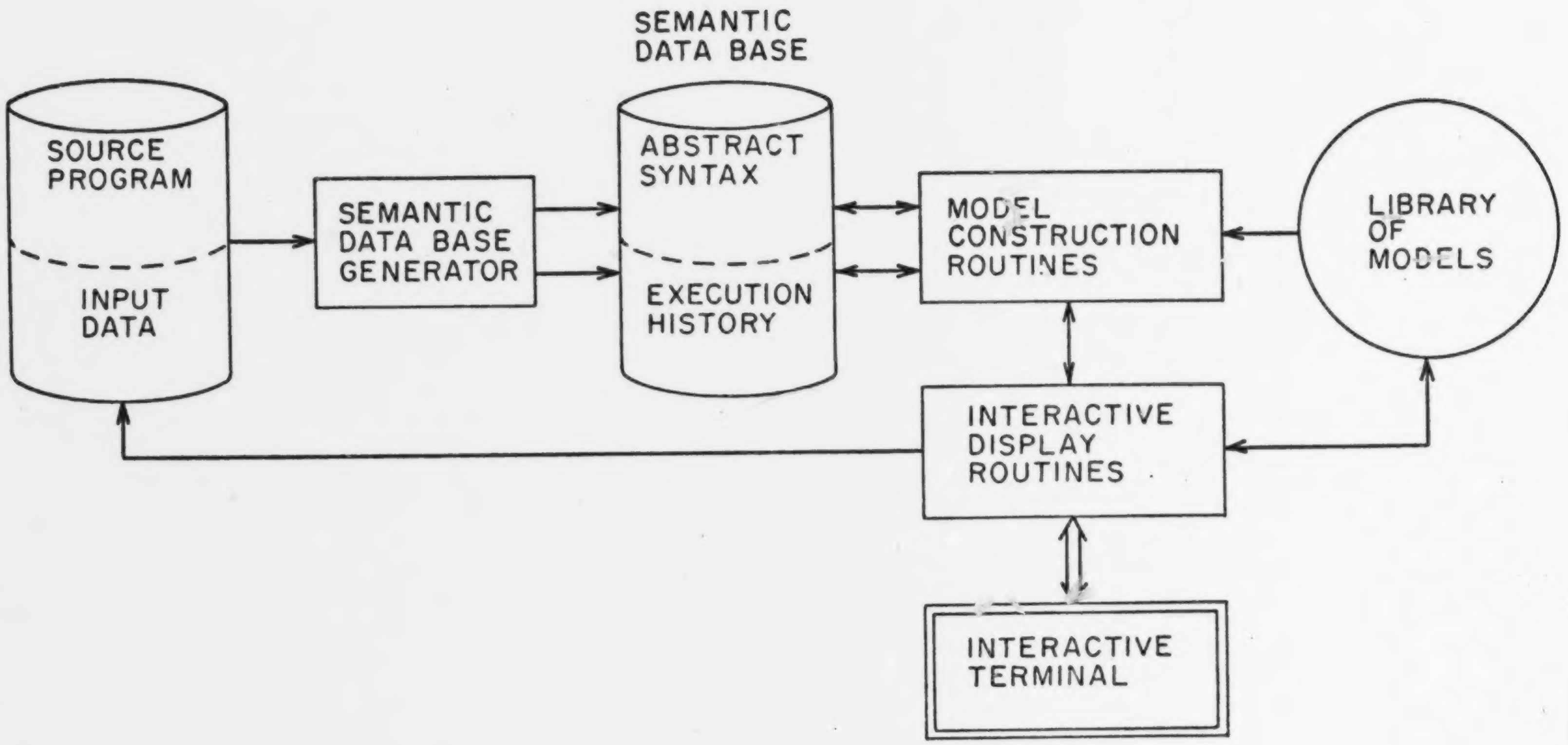
Data sensitivity is sometimes a controversial subject. What I have in mind is to provide an option in the execution of a program to do the execution in artificial arithmetic and to gather information about round off error and truncation error, accumulated from statement to statement. Some of you may be familiar with Herb Bright's work and the package he has produced. That is the kind of thing I have in mind.

Timing information can be gathered by the system in two respects. First, a histogram of statement numbers versus percent of total time spent in that statement (or the number of times the statement was executed). Secondly, we think we can given some reasonably accurate estimates of the actual time spent in the execution of the program. We will compile and record into our data base the number and type of assembly language statements produced by each source statement. Then by using the standard reference manual for the machine to determine the amount of time each assembly language statement takes for execution, we have the information necessary to trace through the actual execution history and add up these numbers to give us an estimate of the total -- the actual execution time for the program.

A couple of other features should be mentioned. The user can provide a Boolean expression (called a "local assertion") describing the relationships among the variables. At that point in execution it will be recorded into the history whether that expression was true or false at the particular time. This gives the user the capability of checking himself on the program logic. Another type of assertion which would seem to be useful is the global assertion which is checked after the program has run. The user can specify, for example, that a variable is monotonic, or that a variable should only have the integer values one through ten. Those kinds of assertions can be checked by looking through the execution history after the program terminates.

SIMULATE and ASSERT are commands devised by Bill Hetzel. SIMULATE is a command which acknowledges the increasing emphasis on structured programming. Using SIMULATE it is possible to simulate the effect of a routine you have not written yet, so that you can call the routine and get back the values that you said that routine would return even though it has not been written. Thus, you can begin checking out the program before certain subroutines are actually written.

Those are the kinds of ideas I am developing, and I certainly am interested in hearing reactions to these ideas. Thank you.


SEMANTIC
DATA BASE
SOURCE
PROGRAM
INPUT
DATA
SEMANTIC
DATA BASE
GENERATOR
ABSTRACT
SYNTAX
EXECUTION
HISTORY
MODEL
CONSTRUCTION
ROUTINES
LIBRARY
OF
MODELS
INTERACTIVE
DISPLAY
ROUTINES
INTERACTIVE
TERMINAL

Schematic Overview of the Semantic Modelling System

Slide RF-1

Two reports by Dorothy E. Lang were distributed to the participants. Titles and abstract follow:

AN EVALUATION OF SUBROUTINE LIBRARIES

Abstract

Some observations are made on the current status of a select few subroutine libraries and packages. The areas of documentation standards, coding conventions, and certification procedures are reviewed and the selected libraries are evaluated with respect to these standards.

DISTRIBUTING SOFTWARE STUDY AND REPORT

Abstract

This paper briefly discusses the pros and cons of distributing software via different media. Included is a preliminary analysis of the costs involved. An attempt is made to evaluate distribution media and draw some conclusions that might suggest possible solutions for the dissemination of mathematical software.

Ms. Lang briefly reviewed the contents of these reports.

QUESTIONS FOR SUBGROUP DISCUSSION OF TOPIC III

1. Are the notions "testing," "portability of programs," and "library development" sufficiently rich conceptually to provide opportunity for publishable research and professional recognition?

2. Is publishable research the greatest need or should we concentrate on gathering and publishing data about various machines and systems?

3. How does research in testing, portability, and library development relate to research in a) programming languages, b) proving correctness of programs, c) numerical analysis and algorithm development?

4. In what areas of testing, portability, and library development do we particularly need models and systematic methodology that would be derived from research?

REPORTS OF SUBGROUP DISCUSSION ON TOPIC III

Leon Osterweil - You recall that Topic III dealt mainly with questions of research in testing and library development. The first question was whether work in the area of testing and library development was publishable work. The first reaction in our group was that the question was vacuous because nobody has tried to publish anything, so nothing was proven. There was an opinion that such work was publishable today but in the future it would become more reputable, and therefore more publishable. There were several people who thought it was questionable today, but likely to be possible in a few years. One person felt that the reverse was perhaps true, that it was more reputable today than it would be in ten years. There was considerable discussion about that because nobody could agree on what constituted an active area of research. Some people felt that a research area could not be construed to be active unless there were many papers being published in that area. Others disagreed completely.

There was a sizable amount of sentiment that research in this area would not be impressive to the academic establishment. This caused concern. At least one person claimed that he would not want to stake his career on it. Maybe I am misquoting to some extent, but his feeling was that he would be uncomfortable in going into this area of research in a heavy way. Several people thought we should compare this research with engineering research rather than mathematical research but this opinion was far from unanimous. In an effort to get at what publishable research really is, one person asserted that publishable work is something that extracts basic principles in such a way that they are applicable to future work. He felt this should be a yardstick and most of the group seemed to agree. There was an opinion, however, that what is most publishable is often merely most fashionable at the time.

We next took up the question of whether publishable research should be the greatest concern of a software alliance as opposed to a concentration on gathering and publishing data about various machines and systems. The opinion was expressed, quite strongly, that the alliance should develop methods and tools. There did not seem to be any real discussion of this and I took that as agreement.

The next question we considered involved a hypothesis with which I personally disagree but I want to report faithfully the group discussion. If we hypothesize that research in portability and library development is not publishable and also that the alliance should work at the development of methods and tools, does this, in fact, dim the prospects for the alliance? In other words, would people be unwilling to join in the work of the alliance if they could not thereby advance their academic careers? Many in the group felt that people would, in fact, work in these areas because they are useful, valuable, and also interesting, even despite the fact that such work might not help them to advance academically.

There was a great deal of enthusiasm for the establishment of a journal of mathematical software. Out of the discussion of such a journal there arose a feeling about publishing which was slightly different from what has been previously expressed. The new sentiment was that work in this area is perhaps unpublishable only because of the current view toward what is publishable research and what is not. The creation of a journal dedicated to this area would, in essence, bring this line of investigation into fashionability, and essentially make it a part of the research establishment. I must hasten to point out, before anybody attacks, that there was a great deal of interest in this journal not just because it would be a forum to achieve academic acceptability, but also because dissemination and awareness were held to be extremely important. The very existence of such a journal would strongly impress upon the various academic communities that there were people working on this line of research, that there was a large community of people who viewed it as viable, and that such work should continue to be pursued.

One other point which was made last night and which seems important is that such a journal could tend to be very ingrown if outside people were not brought into it. In other words, if only people such as the ones in this room were to be interested in the journal and contribute to it, it would not be nearly as effective as if the user community were also involved in it. There seemed to be good agreement on that point.

In summary, the purposes of this journal would be to provide a forum for work in this area, to provide dissemination and awareness, and to be a focal point for work in mathematical software.

Richard Fairley - If there was a consensus in Group B, I will try to contrast it with Group A and also bring up some points that did not come up in Group A.

It was felt that, with notable exceptions, much of the research effort performed in connection with the activities of software testing, portability, and library development does not result in publishable research and the accompanying professional recognition under current editorial and academic policies. It was agreed that creation of a journal devoted to mathematical software would provide a focal point for efforts in the creation, testing, and certification of high-quality mathematical software. Such a journal would facilitate dissemination of information and heighten awareness of work in this area. It might enable important work such as certification of routines to be adequately recognized and stimulated. Moreover, it was agreed that the active participation by the user community would be essential to the continued viability and vitality of the journal.

With respect to the third discussion question, the group felt that research in testing, portability, and library development interfaces with several other areas. In particular, this research should result in recommendations for better programming languages, provide impetus for practical techniques in program correctness proofs and semantic theories of programs, and stimulate dialogue concerning the relationship between machine design and mathematical software. In addition, the research will stimulate activity in numerical analysis and algorithm development.

DISCUSSION TOPIC IV -- USER NEEDS AND MATHEMATICAL SOFTWARE DEVELOPMENT TECHNOLOGY TRANSFER PROJECT; NEED FOR QUALITY SOFTWARE

Roberta Smith - I am very pleased to be part of this discussion as a real live user in your very midst! My talk has three parts. I would like to define our project, then to explain why I feel this definition is necessary, and finally I would like to relate the project to the need for quality mathematical software.

The project which I represent is entitled "Collaborative Research, Computer Based Technology Transfer in Civil Engineering." It is a joint project between the Department of Civil Engineering of Carnegie Mellon University with Steven J. Fenves as principal investigator, the University of Colorado Computer Center with Robert L. Schiffman as principal investigator, and Paul Weidlinger Consulting Engineers with Melvin L. Baron as principal investigator. The study is supported by the N.S.F. from the Engineering Division and the Office of Computing Activities. The purpose of the project is to develop and test general techniques for making application programs and program systems portable and adaptable. I am presenting some of the work being done at the University of Colorado Computing Center by Robert Schiffman, Robert Ewald and myself. Professor Schiffman is also in the Department of Civil Engineering at the University of Colorado. I think this definition is important because we feel we have different problems from projects concerned mainly with defining, developing, and using mathematical software. We also have problems similar to these projects. In particular, we propose that in civil engineering, the major problem with respect to computers at this time is not the quality of the software. The overriding problem is that we cannot use the software which we have. In other words, existing software is not portable. For example, MIT developed a civil engineering-oriented system called ICES which was originally conceived in 1964. One and a half million dollars were spent developing this system on the IBM 360. Control Data wanted to make it operational on the CDC 6000 series machines and estimated that the cost to do so would be $300,000 or 20% of original cost.

Another example, is a large ground shock program called HEMP. It was developed on the Stretch computer at Livermore and is "machine independent." However, ten man-years were required to move the program to the CDC 7600 and it will probably take ten man-years to move it to STAR. This is a significant loss in productivity. I grant that these two examples refer to large systems of programs designed to solve a complicated problem or a set of interdependent problems. More often than not, such programs are machine dependent and more difficult to make portable.

Let me characterize computer software in civil engineering by saying that it may be data based or algorithmically based. If it is data based, it has the whole set of problems peculiar to data based programs, and that is not relevant to this workshop. If it is algorithmically based, its emphasis is on the control string or the engineering algorithm, rather than the mathematical algorithm. The mathematical algorithms are more or less black boxes used by the control string. However, as we see it, these control strings or engineering algorithms are in themselves complicated enough to warrant attention. It took me six months to become convinced of this, but I really am convinced of it now. One example of an engineering algorithm occurs in the problem of slope stability in landslide analysis. There are about eight different methods to analyze this problem. Each one is usually named for the person who first introduced the analytical method. However, the engineer with the given physical situation knows a priori that only 2 or 3 of these methods are applicable to his situation. He would like to use the computer to run a comparative analysis of his problem using these two or three methods.

Another case comes from dam construction and the use of finite element analysis. Here the variables are the dimensions of the problem, and the type of element. Again the engineer wants to be selective. Other examples are soil consolidation programs, where dimensions significantly change the control string and retaining wall analysis programs where the construction may be analyzed for sliding, overturning, or settlement, or any combination of these.

I want to acknowledge that the engineering algorithms in the above examples are not necessary. We could have a separate program for each case. However, there is the feeling that as the huge, complicated system often overpowers the problem to be solved, so also a collection of 10,000 little programs, which cover the basics of civil engineering, is not the answer either. Therefore, the type of program to which we are now directing our attention is in maximum size about 8,000 to 12,000 FORTRAN statements and it runs on a 65K machine. We would like to decompose this program according to engineering assumptions so as to address two problems. First, how can applications programs, which are to be developed, be set up to be portable and adaptable? We feel we can begin to answer this question. A part of the answer certainly is the concept of structured, or top-down programming, which is a major theme, and what everyone seemed to agree upon at the Chapel Hill conference in May. Programming standard, good documentation and users education are a part of the answer. A second question is how can existing programs be made portable and adaptable without redoing them by hand. This is by far the more difficult question. Here we have begun to develop software tools but we are still very much groping in the dark. We want the tools to be portable also, so as a start we decided to develop the programs in ANSI FORTRAN, or as close as we can get to ANSI FORTRAN.

The basic structure of our composition-decomposition system is taken from a generalizer-selector program developed by the NATS project. We plan to build on this system and include in our system the decomposition of the FORTRAN program into logical modules and the identification of machine dependency.

I want to add a word about our view of mathematical software. We want quality mathematical software and we need it. At least at this point in time, we think of it as an obtainable black box. We would especially like these black boxes to be "plugable." By that we mean not only that they are easy to plug into our control string, but also that they go from one machine to another with our programs. We would like the black boxes to be easy to use, well-documented, and return good error messages. It would be nice if they could even anticipate some of our problems in usage through good documentation and error messages. I have in mind that documentation of one subroutine would remind me of the other possibilities within the package of subroutines; perhaps the NATS project already does this. To use a simple example; assume that Subroutine 20 solves the matrix equation AX = B, where A is a banded matrix and a diagonally dominant matrix. I would like the documentation to be specific, and, furthermore, to suggest that if I, the user, know that A is not diagonally dominant (or if I am not sure that it is) then it is better if I use Subroutine 21. Conversely, I would like Subroutine 21 to say that if the matrix has diagonal dominance use Subroutine 20, which will cost you 1/3 or 1/2 of the cost to run.

As a closing comment, I would like to propose the theme "different but not separate", to express our belief that the problems of developing and using quality software in civil engineering are different but not unrelated to the problems of developing and using quality mathematical software.

COMMENTS ON USER/EXPERT RELATIONS

Edward Ng - I was asked to give a few remarks on our experience in user/expert relations, as a stimulant to discussions. It turns out that I am an incomplete expert and imperfect user. My first reaction when I was asked to do this was that there really isn't anything worth saying. So I went and talked to some real experts, in particular, Jim Cody, Fred Krogh, and Chuck Lawson, and I still came away with the feeling that there isn't anything worth saying. So with that in mind, I'll try to keep my presentation to no more than one hour, I promise you! But, seriously, the contribution I hope to make is to summarize several ideas in a large enough perspective so as to provide a systematic beginning to the discussions later.

Those of use who work in mathematical software are trying to automate or simulate some sort of mathematical process which may be numerical, or may involve manipulating algebraic expressions or representing geometric processes. Presumably the experts are the ones who design software for carrying out these processes. Users are the ones who use this software. Most of us here are primarily concerned with the first kind of processes mentioned above. We design our software in two stages. First we try to approximate a continuous problem by a discrete problem, and then we perform the resulting arithmetic on a computer. Naturally an expert may be a user at the same time. In fact, many experts do use software produced by other experts.

Now what in general do the user want? We find that they want a whole variety of things. They do not always come to us with a problem that is very clean cut, like saying they want the number you get by integrating sin x from 0 to 6.4. Quite often they come to us and say "Look at this type of a problem -- Can you given me an idea of how much this will cost me, or whether it's even feasible to approach it at all." Or he might say, "What type of general approach do I need to even start this problem. Should I look into the literature on finite differences or the literature on finite elements, or what?" As we talk further and become more specific, we may finally give some help in the mathematical formulation. Occasionally someone will want an analytic solution "for the sake of publication". Others may want some help in the numerical formulation or some suggestion of numerical methods. Finally there are many more who have completely formulated their problems and all they want are some ready-made computer programs to grind out some numbers. I suppose in this workshop the word "user" mostly refers to this last type.

I have been able to detect six types of attitudes among users. These range from apathetic to superstitious. I find some users rather apathetic to mathematical software for different reasons. First, apathy could come from a user's belief that the construction of mathematical software is a rather trivial matter which he can readily take on himself. Second, it could come from one bad experience that turns a person off. Third, it could come from the difficulty in finding the right software module for a particular problem. You may have heard some user say, "It is easier to write my own than to find the right subroutine for me."

A second type (at the other extreme) is the very ambitious type, who keeps preaching the idea of modifiability. He wants to know exactly what is in the black box so he can tear it apart, and take the stuff that is useful to him. I heard some of this from users who talked at the Purdue meeting.

There is a third class of users whom I call playboys. These are programmers who find their jobs rather dull and who jump at the chance to do something that is halfway mathematical. They prefer to write their own Runge-Kutta routine just for the fun of it. Needing a challenge, they spend their time on things already done better elsewhere.

The fourth class of users I call naive, because they come with problems which are supposedly about mathematical software, but the real problem is their ignorance of FORTRAN, or the operating system, for example.

Then there is the fifth class whom I call the suspicious type. He will take your program and he will do all kinds of testing on it before he can trust anything. Actually, that's fine! In fact, sometimes we encourage it -- at least enough to convince them that our software is good.

The final type, whom I call superstitious, just wants to have a black box that will do exactly what the documentation claims it does. I think the largest class of users that we encounter have this type of attitude.

Of course, these types are not mutually exclusive -- many users have two or three of these attitudes I have distinguished.

Next we turn to the question of what users want in software. Certainly everyone wants and expects reliability in the software he uses. Unfortunately, he does not always get that. Other software characteristics usually wanted include simple documentation, ease of usage, simple diagnostics and efficiency. Some users may also have what I call fancier demands such as modifiability, portability, and interactive capability.

Now let us consider the question of advertising software. We can talk about this on the local (in an organization or a university) or the national level. On the local level, how do we advertise the software that is available? Perhaps readily available write-ups constitute the most important form of advertising. Then we have people -- word of mouth. Tutorial presentations are also used. At JPL we had a series of presentations a couple of years ago in which we called to the attention of many the existence of software. Cody told me that at Argonne they have periodical newsletters that advertise software. Some organizations have so-called "users forums", but they tend to worry about so many things that I have not found them a good medium for advertising software. On the national level the SIGNUM Newsletter is a very important means of communication. Also, we can draw on the experience of IMSL and NATS on their form of advertisement. On the question of a national users group, I think people can sometimes gather in small users groups to discuss one area of problems. As long as it is small, some kind of communication can be established. But when it comes to a larger users group, I think it is hard to imagine focusing on the problems.

Now, what about user/expert interaction? Our first lead to this interaction was through program documentation. That is very important and is why I continue to emphasize the clarity of documentation, in language that is easily understood by engineers. Program diagnostics will cause people to call you. I am not suggesting that we put a lot of bugs in our programs so people will call us! But we should, for example, give reasonable diagnostics so the user can come back to us in abnormal situations. I have recently been a user of a system on the east coast, some 3,000 miles away and we do worry about user/expert interaction. We find that one means of communication that has been quite successful was to have two telephones in your room. One telephone is connected to the terminal and on the other you get consultation. Needless to say, one might have an enormous phone bill in such a case! Fortunately, for many of us, all government labs and many universities have the FTS telephone service, the minimal cost of which makes this kind of thing possible.

I would like to close with one very big question. How should users pay for the experts' service? That is very crucial. I find three levels of support for users. One is the local level, another is the intraorganization level, and the third is the national level. The experts have to be paid from somewhere, somehow. At the local level they draw on overhead from the computing facilities. That puts a lot of pressure on the experts to satisfy short-term local needs, especially if the management is not very sympathetic toward this kind of thing. Users have to worry about their research money and are not always realistic about the cost of expert advice.

On the seond level, I am thinking of a situation where a government agency such as AEC or NASA funds a certain general mathematical software development at some particular place with the idea that the result will benefit other work supported by the same agency.

The third type of support is national in scope, for example, work supported by NSF with the idea that it will eventually be useful in the larger sense for the nation. Personally, I would consider the most comfortable situation for an expert to have about one-half local support and one-half either national or intraorganizational support. This would justify more of your long-term work at a higher level but the local influence would keep you from retreating into an ivory tower. The last remark would have to be modified somewhat for a university professor, but I think the same spirit applies.

QUESTIONS FOR SUBGROUP DISCUSSION OF TOPIC IV

1. Through what mechanism(s) are user needs for mathematical software to come to the attention of software developers?

2. Can a computational mathematician concerned with career growth combine service to the user community with research recognized by his peers? (Obviously some can but what are the ingredients for success?)

3. Through what mechanisms can users and developers work together to influence machine design and software systems?

4. What are the reasonable responsibilities of a computer center upon receiving certified software?

REPORT OF SUBGROUP DISCUSSION OF TOPIC IV

Seldon Stewart - Let me start with the third discussion question which was, "Through what mechanism can users and developers work together to influence machine design and software systems?" Much of the discussion in our group was about the need for users, developers, and a third category -- usually referred to as "experts" -- working together. We felt a need for experts to work with both groups. Development includes the entire range from individual subroutines to large applications packages, and it is important that the experts work closely with the developers so that the best numerical methods can be incorporated into new software. Proper influence by numerical experts early in the implementation will prevent many problems for the users later.

I think there was a very definite expression of the need for missionary work on the part of the experts. They are going to have to go out and tell users and developers about available numerical methods. There was some discussion about users who develop software for their own needs, sometimes without really having the necessary numerical background or knowing where to go for expert help. We discussed briefly the notion of a "hot line," perhaps in conjunction with the software alliance. This would be a manned telephone line that one could call for guidance or help on special areas of software. The response might be to point you to the right kind of specialist to help you with your problems.

We did have a few comments about the choice of languages and their influence on mathematical software, but there was no consensus. Also, everybody was whole-heartedly in favor of computational mathematicians combining service to the user community with their career growth. We discussed several mechanisms by which they would do this but we reached no definite conclusions beyond agreement about the need to combine the two.

There was brief discussion as to what the best interface between users and experts was. The most widely acclaimed interface was through small specialized group meetings which focused on problem areas rather than around professional alignments.

On the final question about responsibilities of computer center receiving certified software, we realized that the computer center does have a responsibility to supply certain kinds of support. We did not decide how much, but they do have a certain responsibility for local testing--not as extensive as required by the certification process, but more than just dumping the package on the system--local maintenance and updating, and their own user servicing. None of these suggestions is unique to mathematical software, except for the kind of testing, which indicates that many conclusions and recommendations about mathematical software might generalize to other software development.

William Hetzel - I will make a brief summary of the points which I thought were significant. Our first question asked how user needs can be made known to software developers. The following points summarize the responses. 1) Feedback from the user/consultant relationship and also through feedback forms; for example, documentation should have pages that can be ripped out and sent back indicating additional needs or problems; 2) Specialized conference sessions to allow selected user interest groups to meet and identify user needs perhaps along the lines of some of the ACM 72 sessions. A particularly important point when a user group got together would be for them to be explicit enough to formulate their needs into a set of specific requirements for new software. In my opinion, one of the better examples of this is the CODASYL experience on data based; 3) Perhaps an expression of user need after the fact would come from usage monitoring of software; 4) One person commented that the mathematical software professional has a responsibility to know the needs of users even if those needs are not explicitly expressed; 5) If an alliance were well established and respected, its reputation would bring out needs; that is, users would feel more confident that expressing their needs would bring results. As it is now, many users are reluctant to express needs and instead write their own software; 6) It was suggested that specific user groups or user representatives might act as sounding boards with software developers who would be encouraged to have interim development sessions that users could attend and critique and hear the impact of their requirements.

There were two comments related to the first question which I thought were important enough to mention. One was that the government is probably the major consumer and, therefore, the most effective one to push forward with standards. Secondly, there are enough examples of hardware that has been tailored to meet express user needs to demonstrate that users can be effective when they are given the responsibility and put forth the effort.

Next we discussed the responsibilities of computer centers receiving certified software. It was generally felt that naming a local representative was the primary responsibility. The degree of his technical competance need not, necessarily, be large but he should function as a two-way channel.

The second responsibility of the receiving site was to publicize the package, announce it, and make documentation available to users. It was felt that suppliers should make this process easy, even to the point of providing newsletter articles. The site has the responsibility of notifying the supplier of problems and relevant experience. The site also has the responsibility to train people in the use of the package.

DISCUSSION TOPIC V -- PUBLICATION OF MATHEMATICAL SOFTWARE

A JOURNAL OF MATHEMATICAL SOFTWARE

John Rice - I am going to be very brief and not try to go into detail. I want to consider general questions about why we should have a journal and how we should go about establishing one. (JR-1) So I am going to list reasons for and against. I am not going to include the standard reasons for the existence of scientific journals. (I assume you know that archives are good things -- good communication is a good thing)

One favorable reason that has been put forth, and I think it is an important one, is the idea of a focal point. You need a place where people who are interested in the creation, analysis, and perfection of mathematical software can focus their attention. I think that the journals of most professional societies do tend to serve as focal points enabling workers to identify with various disciplines and subdisciplines. Mathematical software has the attribute of going in many directions simultaneously without a clear feeling as to what is really going on.

Another purpose that a journal can serve is to set standards. There are standards in almost every level of this whole discussion that have yet to be set, standards in testing, standards in documentation, standards in performance, etc. But we need some place to set those standards.

A professional status function would be served. Those of you who have looked a little at sociology in the scientific community know that there are well-oiled mechanisms for achieving status which vary only slightly from discipline to discipline. One of the standard mechanisms is the professional journal.

A relevant point is the lack of outlet for professional work in mathematical software. You can talk all you want to the deans, to the National Science Foundation, the National Academy of Sciences, etc., about what you do but in the end they want to see standardized stamps of approval.

Those are the four reasons that I have seen motivating such a journal. I see three drawbacks to publishing a journal -- nontrivial ones, but very mundane. It takes work, it takes money, it takes negotiations. The negotiations come from the fact that we do not have standards in this area and therefore if everyone here wrote a proposal for what that journal should do there probably would be no more than 50% intersection between any two proposals. There is a lot of work involved in finding reviewers and getting set up.

If, however, one decides to go ahead and try to create a journal, then I think there are three sorts of "how" questions. The first of these is about a professional home for a journal. A home is needed for stability. You must have somebody who knows what is happening on a continuing basis, somebody who knows how to get galley proofs back and forth, copies sent out and all that sort of thing. A professional society is one possibility for the field of mathematical software and there are other possibilities. National laboratories publish journals. Universities publish journals. And there is what might be called a professionally independent home, in which the stable center is a commercial publisher. All the operational personnel on the professional levels can move at will and the focal point of the production process is commercial publishing.

The second formal question involves coming to grips with the editorial policy. We must ask about the scope of this journal. We see discussion at the meeting about the scope of an alliance. I think that the scope of mathematical software includes lots of things that have not been mentioned here, and on the other hand, some things that have been discussed here probably would not fall into the scope of a journal. So it is something that would have to be worked out.

You have to try to establish standards. Not having many real examples of the kind of paper that you might want to publish you would have to say what you expect from papers that are submitted. Finally, you must work out refereeing procedures and I am sure Lloyd Fosdick will tell you that is a nontrivial process for mathematical software compared to the standardized scheme of the typical journal where the editor sends out papers and asks for opinion. The third type of question is about actual production policy and problems: How much is it going to cost? How frequently? How big? How are you going to publish programs? All of these things have to be decided.

I think I will conclude with my opinion as to what the next step would be if people decide to go ahead with these ideas. I think that a relatively small group of people must come up with a concrete proposal for what this journal is going to be, how it is going to operate and what its professional home might be. I think a small group is essential there. We have already seen the problem of defining "certification" and I suppose we could be here a month trying to hammer out a consensus on the scope of the journal, for example. Even three people might take a long time. Some small group has to attempt if it is going to be done with any efficiency.

JOURNAL OF MATHEMATICAL SOFTWARE

Two General Questions: Why & How

"Why" Reasons for (besides the standard ones)

A. Focal Point - creation, analysis, perfection
B. Standards - performance, analysis, programming & documentation
C. Professional Status - for contributors
D. Outlet - now lacking to a varying extent

Reasons against

A. Work
B. Cost
C. Negotiations on divergent viewpoints

"How"

1. Professional Home
   Society: ACM, SIGNUM, collection of SIG's
   Institution: University, National Laboratory
   Independent: Commercial publisher
2. Editorial Policies
   Scope
   Standards
   Refereeing procedures
3. Production Policy & Problems
   Cost, Frequency, Size, Program Manipulation

Opinion: "How" list is in order of decreasing difficulty and of increasing effort requirements.

First Step: Group (3-5) must come up with reasonably detailed proposal and attack the "professional home" problem.

Slide JR-1

DISCUSSION TOPIC VI -- ORGANIZATION TO FOSTER MATHEMATICAL SOFTWARE DEVELOPMENT

A MATHEMATICAL SOFTWARE ALLIANCE

Wayne Cowell - A few weeks ago when we sent out the invitations to this workshop we distributed a working paper entitled, "A Mathematical Software Alliance," dated June 5, 1972. I would like to review this paper briefly, giving some of the highlights in it so as to establish the considerations which we feel are important.

Let me begin by showing a slide I showed the other day (WC-1). In order to achieve the goal of producing systemetized packages there are many steps that one has to take. The basic mathematical analysis serves as a foundation for the development of numerical methods which in turn result in algorithms which might be expressed in Algol, for example. From these algorithms we write computer programs. Much of what we have discussed at this workshop is the process of making those programs into systemetized collections This is a rather long chain of events and some of these steps are very involved and require a great deal of various kinds of talent. With extremely rare exceptions, we do not expect to find in any one person or in any one institution the talent to start at the beginning of this process and carry it all the way through. For that reason, we are talking in terms of a software alliance whereby a group of institutions and individuals would form a consortium to carry out the necessary steps.

Another way of expressing this chain of events is to identify three types of activities: (1) Research and implementation, (2) evaluation and refinement, (3) dissemination and support. To make this somewhat more tangible let me use EISPACK as an example. The research and implementation for EISPACK was all the work over many years leading to the Algol versions as published in Wilkinson and Reinsch plus the additional step of translating the Algol into Fortran. Evaluation and refinement was the field testing on various machines as carried out by the NATS project. The dissemination and support is the activity of distributing these routines from the Argonne Code Center and supporting them as specified in the statement of certification.

For the most part, present day efforts to provide mathematical software are not integrated over the chain of events described above. I believe you would be able to find all of the steps going on and in some cases there has been an attempt to tie them together. But there has not really been an organized attempt to focus on the process itself -- the whole process. We could talk in terms of forming some sort of comprehensive mathematical software institute in which all the types of talent would be concentrated. I think that is an unrealistic extreme. A viable middle-ground approach is the concept of an alliance to focus the effort. In the working paper we give a number of examples of activities appropriate to the three stages. It is not meant to be an exhaustive list. I am sure you can add to it. Some of you might want to modify some of the items on it.

Now I would like to say some specific things to set the stage for further discussion. I do not believe that this is a group where we can actually engage in detailed planning of exactly how we are going to organize a new structure. I do think, however, that we can attempt to find a sense of the general direction we want to go. Assuming we are successful in that then I believe that the following specific steps should be taken. In the first place, there will be a report of this workshop. We have been using the tape recorder and we will transcribe this material, edit it, and include it in a report to the National Science Foundation. Following that, Lloyd and I expect to write a joint proposal for the formation of an advisory panel such as is described on pages 5 and 6 of the working paper. This would be a group of six to twelve persons who are recognized and respected authorities in mathematical software. This group would give expert technical guidance in the choice of particular mathematical software objectives to pursue as tasks of an alliance. Hopefully, this would stimulate proposals to carry out these tasks. The group of experts would continue to review the objectives and evaluate progress toward them. The advisory panel may decide to form an executive board of people who are going to meet rather frequently and do some concentrated work. This is the approach advocated in the working paper.

The advisory panel could begin by reviewing and assuming policy guidance for two activities now being proposed which will likely be underway by the time the advisory panel is formed. The first of these is NATS II which is a joint proposal from Argonne, the University of Kentucky, and JPL. To build on our experience with NATS we propose to carry out a similar set of activities for various routines. We have added a research and implementation component and have made an attempt to find a mechanism for paying the test sites for some of their activities.

The second activity has been proposed by Lloyd Fosdick and is concerned with the analysis of mathematical software. He discussed his approach on the first day of the workshop. The rational in asking the advisory panel to look at these activities is that both of them would appropriately take place under the auspices of a mathematical software alliance if there were one.

The advisory panel and executive board would also be concerned with further development of an alliance by seeking to establish centers where certain types of activity would be focused. As you can see from the working paper we have tended to associate research and implementation with the university setting, evaluation and refinement with the national laboratory setting and dissemination and support with the private sector. Because of the people who are involved, the University of Colorado and Argonne National Laboratory are natural candidates for the university and national laboratory. But, of course, that is open to further discussion.

REMARKS ON IMSL LIBRARY ACTIVITIES

Edward Battiste - I want to thank Wayne, Lloyd and Gordon for having us here. We are one of the few private concerns in attendance, and our area of interest is exactly the area of interest of this group. I was asked to talk about IMSL activities. They are all well explained in brochures which are laid out here and you are welcome to copies. (Please do not take the order blanks. Those are reserved for those attending organizations which have not subscribed.) In my discussion, I will make some points pertinent to the alliance, then tell you where IMSL is today, and what we intend to do. Also, I will remark on several points made by previous speakers.

My remarks pertinent to the alliance are stated in the form of enigmas. The list is not exhaustive; however, I feel that these points certainly should be considered as we more towards an alliance-type organization.

1. The "Committee" Enigma

At a certain point in software development, committees become a vehicle for discouraging the application of effort in the timeframe necessary for completion of the detailed productive task. That point is reached when software is being developed for dissemination, as opposed to development for examination, of algorithms. This problem is especially critical because everything in computing evolves so rapidly. The committee that developed NATS (if it was a committee) was well ruled; NATS did distribute very good work. Generally, committees cannot make decisions in the detail required for production in a timeframe which considers that software is needed by users before its base of usage (languages and computers) dies.

2. The "Expert" Enigma

One might think that experts were not available in the desired quantity to an organization that had the intent of producing good software. We asked only eleven advisors to join us. They were asked because we knew them and knew of their work. Not all of those needed were asked. But, all eleven joined and took the part that we asked them to take. Monetary gain was apparently not involved in their decision. Experts are available, and they will be available to the alliance also. However, people who have been involved in library development know that people are not available to apply control to development over a long time period. How do we evolve a system which allows expert input, continuity, and continuous control, for software development (not algorithmic development, but software development)?

3. The "Do-It-Right-Once" Enigma

Many people have started library development. Many people know someting about the subject. Some may still feel that it is possible to do it right once (at last). Beginnings are easy. Long-range ideas are very difficult to bring to fruition. Mechanisms for evoluation must be built into library development processes. These must allow usage during evolvement.

4. The "Software is Inexpensive" Enigma

Software certainly won't be inexpensive for the alliance to produce, and it is not inexpensive for IMSL to produce. Our costs, for intial development, ranged from $600 to $6,000 per code, up to the point at which maintenance began. These costs were minimal. I do not believe we wasted money; we do not have an organization that can allow waste. Also, we really operated more as a benevolent dictatorship than as a committee. Some aspects of software development require extreme orderliness. Our advisors aided us by giving us the input asked for and by not really dictating to us. This was correct, because a development group can only take input of the type that is available from a talented group of advisors in the order in which it can be implemented. However, during marketing, we were told several times, by very experienced computing specialists, that a lease price of $720 per year for 250 subroutines (which cost $375,000 to produce) was much too high.

5. The "Control is no Problem" Enigma

People (those who have not built libraries, for example) seem to feel that the task of obtaining a correct, running, tested program is the major problem in software development. Control of the development, testing, testing maintenance, document evoluation, personnel interaction, and code maintenance of a growing, interleaved set of abilities, is the whole problem. Obtaining one well programmed code is no problem at all. Control is no problem if it is not attempted.

6. The "Profit Motive" Enigma

(Someday we hope to have a profit motive!) Maintaining continuity of effort in a detailed software (not algorithm) development, is difficult. Several points can be made. First, each person must have knowledge that, with the resources at his command, he has the ability to do the job well -- each person, not just the designers and the "thinkers." Second, each person has to have a horizon which allows him to feel that movement of details to other shoulders, some day in the future, may be feasible. These shoulders can be "human shoulders" or "automation shoulders". Third, this effort requires control which is implied by a slightly dictatorial system during the development phase. The efficiency dictated by an organization which relates its efforts to providing good results for potential monetary gain is probably necessary. A profit motive does not mean that research will not take place. I do not understand that as relating to the profit motive in any nonpositive manner.

7. The "Giant-Step-Forward" Enigma

The fast evolutionary nature of this industry demands small steps in order that good work reach its audience at all. The pertinent audience is not "research" or "academia." Most of the scientific computing in the world in done in industry. That is the audience that needs our work.

8. The "Cerfitication" Enigma (This is not really an enigma)

The process of certification implies an expansion of the required timeframe that is worrisome. We should not take too long to certify codes; certainly everything cannot be certified. The selection process for codes which are to be certified must be a good one.

9. The "Graduate Student" Enigma

IMSL needs good programmer-analysts. These are programmers who have been put past a variety of scientific computing tasks many times. I disagree that development can currently take place in universities. By development, I mean development of codes that are tailored rather precisely to give good results. This situation may change, but change will take a long time, and, at the moment, I do not see the university as being able to disseminate codes to the proper audience unless the certification process is quite good. I do not believe that a graduate student can be a good programmer analyst for this type of development work. They have other desires, and time requirements, and good programmer analysts need much well-directed training.

That completes the "enigma" discussion. Those "enigmas" are a part of the reason for IMSL's entry on the scientific computing scene.

What is IMSL doing today? We have released three libraries, which we keep parallel in content and ability. Most of the people who came with us came from IBM. Thus, we developed our first library on IBM equipment, using assembly language at times. For our other libraries, we do not have the resources to continue that (assembly language) attitude. When we map our libraries we map them in FORTRAN. Quite often we change the algorithm in mapping, but more often than not the algorithm remains the same.

Our advisors: Here's what they do for us now. They direct us to good algorithms. They answer our questions when we ask them, and they help us in marketing by mentioning our work. Sometimes they test for us. We hope that our relationship with the advisors is in concert with their interests and time constraints.

Testing: We do all testing "in-house." We have documented standards which are evolving and which have not yet been published. In our testing, we consider the various algorithms used, and test over their ranges, against references or long precision versions of the codes. We are especially careful at "cross-over points". We test all basic blocks, and perturb all the logic (although not in the depth that Lloyd has discussed at this workshop). We perturb all error indicators. Most of our initial maintenance letters (noting code changes) were generated by IMSL. We required that our own people build applications programs using the library; this generated code changes. We obtain a great deal of good information from our users -- the University of Colorado pointed out an error recently referring to a situation that was not discussed in the literature and that was not detected in testing. By and large, maintenance changes have been no problem, but many are still generated by us.

Our original intent was to build a set of kernels which span, in some manner, mathematics and statistics. We included evolution (new editions) because we knew the set could not be perfect at any given time. We hope that this basic set of libraries will pay our expenses. On this base, as we grow, we intend to build application programs. Many industrial organizations do not like the task of picking out the set of kernels that go together to solve even a simple problem. Application programs will be our second level of endeavor; we are just embarking on these. Our third level will be to take chapters of codes which we think are adequate and to put them into an environment for access by workers in a particular field, perhaps interactively.

We would like to be able to study portability and hope to have that opportunity. One speaker mentioned that portability was of utmost importance. From the point of view of a user in an industrial environment, there is only one thing that is of the utmost importance -- a correct answer. If one contemplates or philosophizes in an industrial environment and spends a year on "how a code can be made portable," nothing is accomplished. There will be no answers (and there will soon be no group). In present company the topic is a valid topic.

Our problems: One of our problems is the marketplace. Unfortunately, we have to market this product. There are no salesmen that are trained to do this, so that we are deeply involved in marketing the product. Fortunately, "hard sell" is not required.

We have some personnel problems. Some personnel do not like the control requirement for production of a product which run on computers. Others do not like chaos. We are handling that problem fairly successfully.

A major problem in software development is clerical in nature. We need a giant system. Right now we use 195's. Card handling is a terrible task. Everyone needs a desk terminal in our environment.

Another minor problem is that in library development statistics is different from mathematics. Note that our advisors are heavily weighted in favor of numerical mathematicians. Not many statisticians are really interested in this topic. Also, statistics does not "kernelize" as well as mathematics (there are some chapter exceptions). This is a problem because our library is not only a numerical analysis library.

Our three major problems are:

1. The automation of documentation, both for our development, and for use by our subscribers. We think the Argonne documents are excellent.

2. Portability: We want to produce one library.

3. Testing - both mechanisms and maintenance of test codes. Our accuracy statements must be upgraded. We maintain, now, 800 programs and 1200 test codes.

Because of the need to investigate these areas, we do not consider that a library structure change will be possible before 1976.

Further comments on the alliance: First, I feel that I should alter some statements which are before you in the second working paper. IMSL does not maintain that more than one level of quality is acceptable. We feel that users need good software and that the timeframe required for bringing a broad range of software to one level of perfection is quite long. On our approach to certification: Our advisors do not certify our codes.

Second, and finally: I do not believe that there is any organization on earth, public or private, that can support a code which it did not develop. The alliance may be able to do so at some future time, maybe some ideas which were mentioned here will allow such support. Support is a very "tough topic", as you know, and IMSL feels, today, that it only can support those codes which it produces.

Thank you again for inviting us and for allowing us to present our ideas.

NSF NETWORK, ACTIVITIES AND PLANS

Gordon Sherman - As you all know, the chief interests and purposes of the National Science Foundation are in supporting basic research and improving the general health of scientific research in our nation. In particular, NSF has supported basic research work in modern computing, computer science, computer usage in education, and also different projects designed to make computing facilities more available to researchers, particularly those in large universities. Over the last several years the Foundation has supported much work in networking. The computerized regional education project is one of our major efforts but, in addition, the NSF has supported many other projects which have been aimed at bettering the practical knowhow as well as the technology of networking. The emphasis has been on sharing scarce computer resources.

Recently, the Foundation through the Office of Computer Activities, has brought all these varied programs together under one program. It's an expanded research program that could lead to the development of a national science computer network. Such a network would link colleges and universities and other institutions in national support of computer oriented research and education. There are many complex problems involved in constructing a network, and in particular, constructing a large national network as we now envision. The tough problems aren't all technical by any means. Many believe that the most important problems have to do with the management of networks, financing them, and scheduling the use of hardware and data banks and making necessary software available. The advantages, I think should be clear to almost anyone and could have a positive effect on other current research interests. For example, another program we have in the Foundation has to do with improving the quality of software and particularly portability of software. One of the things we have found out, as exemplified by the NATS project, is that quality software is very expensive. It is far more expensive than was formerly believed. If large networks were constructed and were successful, it would go a long ways toward helping to solve the problems we now face in the portability of software and the availability of high quality software.

Computer graphics is another example. Many new and interesting things are developing in the application of computer graphics. It's a wide open area, and there are many useful applications that are yet to be uncovered. But much of computer graphics is expensive and doesn't even exist at many places. A good working network could be extremely valuable in providing computer graphics to more researchers.

With respect to the newly announced networking program, I'd say that right now the main objective, as I see it anyway, is to visualize beforehand as many of the potential problems of implementing such a network as we can. That is, before actually constructing this network we want to identify some of the problems which would crop up, to foresee some important things, and to do some investigation and experimentation. We want to do all we can to plan a network and do it well. Another objective, I would say, would be to help justify the cost. The resources which it would take from federal funds must be justified if it is decided sometime in the future to actually implement such a network. We are looking for help from qualified computer scientists and interested people. If you have any ideas for a project which you would like to do and which may have a bearing on the subject, people at the Foundation would be interested in talking to you.

The program is a National Science Foundation program, and it involves several offices at the NSF. The Office of Computer Activities is the coordinator of the program, but also the Office of Science Information Services is very much involved and has supplied some of the budget money to support the program. Dr. D. Don Aufenkamp of the OCA/NSF is the program director. There is also interest in computer networks within other programs in the Foundation, for example, mathematics, chemistry, biology, and social science. If, for example, some aspect of a proposal involves mathematics but also has a bearing on the National Science Network, it may be that both OCA and the Math Sciences division of NSF could be involved.

We're just entering the planning stage. I have a limited number of descriptive brochures here which I'll pass out.

QUESTIONS FOR SUBGROUP DISCUSSION OF TOPIC VI

1. A basic assertion in the working paper on a mathematical software alliance is that "the present situation in mathematical software is unsatisfactory and new approaches are required." Is this a tenable position?

2. The working paper takes as a premise that "an alliance of institutions is needed to provide the necessary range of talent to produce, evaluate, and disseminate mathematical software." Is this premise justifiable? (Presumably the alternatives are "more of what we now have" or a centralized "institute for mathematical software.')

3. The working paper asserts that mathematical software evolves through three stages. The software alliance organization is based on this assertion and associates Stage I with universities, Stage II with government laboratories and Stage III with the private sector. Are these stages a valid model of the development process? Is the pairing with institutions meaningful? Is the pairing oversimplfied?

4. How can a computer network be utilized in testing mathematical software?; in disseminating mathematical software?

5. Assuming that the answers to the first three questions are basically affirmative, is the Advisory Panel/Executive Board approach described in the working paper a viable way to establish and administer the alliance?

6. Assuming basically affirmative answers above, which of the four plans presented in the working paper appears to offer the best hope of successfully meeting the objectives? What alternative plans as a substitute for or modification of the four do you suggest? Please document the answer to this question, especially by commenting on the arguments given in the working paper.

6'. If any of questions 1-3 or 5 above were answered negatively and alternatives proposed then this constitutes a request for comparisons with the approaches in the working paper.

7. On page 11 of the working paper under "comments on the involvement of the private sector" tentative conclusions are reached concerning the distribution of costs. Are these conclusions valid, partly valid, or false?

8. Cowell and Fosdick estimate that the annual operating cost of a mathematical software alliance would begin at about 500K and stabilize at around 1M. Are these figures high, low, or about right?

REPORTS OF SUBGROUP DISCUSSION OF TOPIC VI

Stuart Lynn - On question one, the assertion that the present situation in mathematical software is unsatisfactory, I think there was very little discussion on that. I think we all agreed it was a tenable position. That's why we're here.

The second question was somewhat more extensively examined. It inquired as to whether an alliance of institutions is indeed necessary to start attacking these problems. We did conclude that such an alliance was necessary, even though we didn't fully understand the details of what such an alliance would look like. That was discussed later. We discussed that fact that perhaps NATS and the upcoming NATS II may be somewhat typical of what might be expected. The principle difference between the NATS projects and the proposed alliance is that the alliance would probably have more continuity and permanence and it might be able to subcontract or at least be able to advise on subcontracting money. There was some concern expressed that the alliance might be exclusionary in that it might only direct funds toward those institutions that are involved with the alliance and exclude others. Hopefully steps would be taken to assure that that would not be the case.

There was a strong concern about the fact that the alliance as discussed and described does not attack the important problem of user recognition of this area that we think is so serious, and establishing that it is indeed as serious and important to them as it is to us. It was clear to the group that there would be a need for the alliance to be very actively engaged in attacking this problem through various techniques of marketing, through efforts to drive out bad software one way or another. There was some concern that such software might not be so easily driven out.

There was a lot of discussion on precisely what the prime objectives of an alliance are. Are they principally concerned with methodology of testing and certifying and producing software, or is the prime concern of the alliance to be with the actual production and dissemination of software. I think it was generally felt that production and dissemination were objectives that should assume much lower priority in the alliance than the objective of evaluation of existing software. It wasn't clear to the group whether it could effectively engage in production and dissemination. This is not to say that these activities should be excluded entirely because the prime objective of studying methodology must be served by becoming involved in problems associated with production and dissemination as well as evaluation. One point of view was that we may lose our credibility if we gain the reputation of trying to push our own products just because we produced them. This conflicts with objective evaluation of other products. We reached the consensus that an alliance is needed in some form. Primarily it should focus on questions relating to evaluation and methodology. Production and dissemination are far less certain as objectives requiring far more careful consideration in the future before embarking too heavily in those directions.

The next question concerned whether or not the three box model was a valid model or whether it was an oversimplified model. The model as described was somewhat oversimplified; basically the labels in the boxes were indeed correct but the flow between the names might not be correct. Many alternative flows would have to be considered and many types of feedback might occur between these areas. It was also expressed that careful consideration needs to be given to insuring that there is indeed a flow of people between these various areas of responsibility. There should be a constant chain of people who become familiar with all the various problems of all the various areas in order to keep people excited and motivated.

We then started to examine the question of whether the institutional pairing proposed was indeed a valid pairing. I think we rapidly established the viewpoint that it was sufficient but not necessary. There are many other alternative pairings which might be considered. There was also concern expressed about the role of universities in implementation of software, whether this is indeed the principle role that universities should be playing.

We skipped the question on network utilization because we felt that it wasn't in the mainstream of what we were trying to do in this session.

We had a long discussion on the question of the viability of this advisory board and executive committee structure for the alliance. We felt that the responsibilities as outlined in the reports were not sufficiently proscribed to permit definitive statements as to whether it's the most suitable approach to take. We felt that the advisory board was presented in a way that confused two roles which tend to be distinct. One is the technical advisory role and the other is the role concerned with responsibility for the establishment of policy and planning, general coordination, and management. These two roles are logically different and perhaps should be separated in the future. After a wide-ranging discussion of technical responsibility and responsibility to the funding agency, we reached a consensus that a proposal to the NSF should be made to establish that we named an advisory council. We never established how people should be appointed to this council but we felt it should have wide representation to engage in the detailed preliminary planning and establishing of policy for an alliance of those concerned with mathematical software. Among other things the council should carefully define the administrative structure of an alliance, paying attention to the following three components: (1) the technical advisory component, (2) the policy and planning component, and (3) the operational component. The council should carefully examine how the proposed administrative structure would interact with other boards and with NSF.

As I indicated before, we never decided precisely how we were going to establish this council, but we did indicate that it should have representatives from users of mathematical software and from the private sector as well as experts on mathematical software.

We did raise the question as to whether there were any viable alternatives to the proposed approach. One alternative that was suggested was a non-profit foundation instead of the proposed council. There were certain advantages to that since it was felt that the administrative structure would be much more clearly defined but there was a question as to whether anybody would want to pursue this approach.

At this point we did not attempt to look at various administrative structures since we thought this was something the council could come to grips with. But we did point out that we thought the correct approach was for the council to decide what it was going to do, and then how it was going to do it. In other words, structure should follow much later in the planning.

A couple of additional comments are worth reporting. It was remarked that if production was something that the alliance did engage in there should be a charge for the software for several reasons: one is to be able to relate more fairly to the private sector. Another reason was to weed out the guys who want anything for free. You tend to use a product that you pay for.

We felt that questions of funding would follow later. We didn't feel we had enough information to say what level of fuding was appropriate at this time. Again it's a question of deciding what we're going to do, then working backward. It is important that priorities be established and money be spent wisely. I think that concludes what I have in my notes.

Charles Lawson - I think Stuart has simplified my job. That was very well organized. I am happy to say he hit many of the same points we talked about. I will try to fill in where I think we may have had something additional or different to say.

On the first question, we did expand a little more on whether or not we have a problem at the moment. The first thing that comes to mind is whether one could say that a major project might fail due to bad mathematical software; but in fact, no one in our group seemed to be able to come up with specific examples as extreme as that. The more usual problem is that unreliable software is costly because valuable resources are wasted. Users try different subroutines or write their own. The computing facility has to store and keep track of an excessive number of subroutines because they do not know which ones they could throw away. Also, there is the feeling that there's a real need for better program development methods, especially if future programs are anticipated to be more complex than present ones.

We got into several controversies on question two. One of them looked like the following. Given that the main problem is the excessive number of bad subroutines a direct attack on the problem would be to try to identify these bad subroutines. This would imply that an important mission of an alliance would be to test all available subroutines of a certain type and identify the good and bad ones. Computing facilities could then get rid of the bad ones. This would imply that the alliance should concentrate on evaluation rather than production. Part of this viewpoint is the tendency to think of evaluation as an elaborate process as witnessed by our debates over the definition of certification. But it was brought up in our discussion that the identification of bad software is not necessarily that elaborate a process. The paper by Wampler of NBS was mentioned as an example of testing that used only two test cases. But it was performed and reported very carefully and the tests were reproducible. It gave obviously bad marks to quite a few routines and according to Hans Oser, the National Bureau of Standards has received many inquiries about this paper. They feel it has had a big impact. I do not think we were saying that evaluation is always simple but neither does it always require the full force of a lot of experts to eliminate inadequate codes.

The other view is that it is cheaper to have an expert produce a good subroutine than to evaluate all existing subroutines. An example of this would be the NATS project where there was no effort to conduct a survey. There was just a feeling among certain experts that the Wilkinson codes represented the best way to do it and so they went in and did it.

Another controversy centered on the separation of production and evaluation. One view is that such production as is done by the alliance should be done by different people than are doing the evaluation and preferably under different roofs. This would follow the practices of some large commercial software companies that have found this is the most effective way to produce reliable software. Those holding a different view pointed out that the NATS project has not so far separated production from evaluation, and that past history does not show enthusiasm or competence to evaluate mathematical software by persons who are not involved in its production. However, the EISPACK experience may not be typical in the sense that there was a code ripe for harvesting, and that exact parallel situation may not come up again.

Question three asked whether the three stages of the working paper constitute a reasonable model. It was suggested that one might identify five stages, namely, algorithmic development, systemized packaging, evaluation, distribution, support. I think the key thing here is separating distribution and support because there seemed to be a feeling in our group that support has to be tied closely with either evaluation or development. The people who give answers of a supporting nature are the ones that have done either the development or the evaluation. Also the people who do the evaluation are interested in feedback for their own professional interest or better understanding of the problem. Are the pairings with institutions meaningful? Well, the pairings were fairly arbitrary and might be okay as an administrative umbrella. But it was pointed out that the NATS experience so far depended on results developed by Wilkinson and by Cody, neither of whom were in a university. So the idea that development belongs in a university still is an abstract notion, rather than reflecting how things have been done. Again, in a separation of tasks, we felt that distribution by itself is probably not a commercially attractive enterprise.

We saw question four the same as the other group. No comment.

On question five, the advisory panel-executive board structure was accepted unanimously by our group. It seemed to be a major point of discussion in the other group. However, later discussion did bring up the question whether the advisory panel would include people outside the numerical analysis community, for example, statisticians.

On the budget question, we pretty much accepted what was suggested although this matter of the importance of production was discussed quite a bit in terms of budget. Some people felt that not more than 10% of the budget should go into production because the main unique impact of this organization would probably be in evaluation.

Now that finishes my report on the questions as listed, but we got into a terminal discussion that took us back to the goals of the organization. These were broadly seen as research and application in areas of software development that would have an impact on real problems. It was felt that the research in the areas of testing methodology and programming methodology were probably more significant things to shoot for than the area of algorithmic development. This would also be there, but is not quite so uniquely associated with this organization. The organization should stimulate research in other areas than just what it was doing itself. My own thought is that the link between research and the user is one of the more unique things about this organization. Research does go on right now, but it does not get pulled all the way through to the users. Applications are surely handled, because a person can go out and contract for somebody to write him a differential equation subroutine and get it written. But the unique thing about this alliance is trying to carry a specific research effort all the way through to the end users.

A couple more miscellaneous remarks. I think our feeling was that the private sector did not fit into one box the way it was shown in the diagram. Some studies have shown that private companies can do a more economical job of software production than a university computing facility can. So, maybe the production end should be farmed out to the private sector in some cases. So the involvement of the private sector should be kept more flexible, and used where it is more appropriate. Also, it seemed like centering things at two places was probably not terribly appropriate. But this ties in with what Stuart said; after the goals are better formed, the structure will follow.

LIST OF ATTENDEES

Thomas Aird
Purdue University

Edward Battiste
International Mathematical &
Statistical Libraries, Inc.

James Boyle
Argonne National Laboratory

Wayne Cowell
Argonne National Laboratory

Augustin Dubrulle
International Business
Machines Corp.

Richard Fairley
University of Colorado

Lloyd Fosdick
University of Colorado

Fred Fritsch
Lawrence Livermore Laboratory

William Hetzel
University of North Carolina

Thomas Hull
University of Toronto

Yasuhiko Ikebe
University of Texas

Fred Krogh
Jet Propulsion Laboratory

Dorothy Lang
University of Colorado (student)

Charles Lawson
Jet Propulsion Laboratory

M. Stuart Lynn
Rice University

Cleve Moler
University of New Mexico

A. C. R. Newbery
University of Kentucky

Edward Ng
Jet Propulsion Laboratory

Hans Oser
National Bureau of Standards

Leon Osterweil
University of Colorado

John Rice
Purdue University

Walter Sadowski
National Bureau of Standards

Gordon Sherman
National Science Foundation

Lyle Smith
University of Colorado

Roberta Smith
University of Colorado

Seldon Stewart
National Bureau of Standards

Henry Thacher
University of Kentucky

Joseph Traub
Carnegie-Mellon University

Jeffrey Wright
University of Colorado (student)

Jacob Wu
University of Colorado (student)

David Young
University of Texas